ESD-TR-66-94

MTR-122

AD 642331

XPSTC, A FORTRAN DYNAMIC STORAGE ALLOCATOR

OCTOBER 1966

J. E. Sullivan

Prepared for
DEPUTY FOR ENGINEERING AND TECHNOLOGY
COMPUTER PROGRAMMING DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

CLEARINGHOUSE
FOR FEDERAL SCIENTIFIC AND
TECHNICAL INFORMATION

Hardcopy	Microfiche		
$ 3.00	$ .65	77 pp	as

1 ARCHIVE COPY

DDC
RECEIVED
NOV 28 1966

Distribution of this document is unlimited.

Project 250G

Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts

Contract AF19(628)-5165

ESD-TR-66-94
ESTI FILE COPY

ESD-TR-66-94

MTR-122

ESD RECORD COPY
RETURN TO
SCIENTIFIC & TECHNICAL INFORMATION DIVISION
(ESTI), BUILDING 1211

ESD ACCESSION LIST
ESTI Call No. AL 5342?
Copy No. 1 of 3 cys.

XPSTC, A FORTRAN DYNAMIC STORAGE ALLOCATOR

OCTOBER 1966

J. E. Sullivan

Prepared for
DEPUTY FOR ENGINEERING AND TECHNOLOGY
COMPUTER PROGRAMMING DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 250G

Distribution of this document is unlimited.

Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts

Contract AF19(628)-5165

AD642331

This document may be reproduced to satisfy official needs of U.S. Government agencies. No other reproduction authorized except with permission of Hq. Electronic Systems Division, ATTN: ESTI.

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

ESD-TR-66-94

MTR-122

XPSTC, A FORTRAN DYNAMIC STORAGE ALLOCATOR

OCTOBER 1966

J. E. Sullivan

Prepared for
DEPUTY FOR ENGINEERING AND TECHNOLOGY
COMPUTER PROGRAMMING DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Distribution of this document is unlimited.

Project 250G

Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts

Contract AF19(628)-5165

ABSTRACT

The usage and other characteristics of XPSTC, a FORTRAN - compatability storage allocator for the IBM 7030, are discussed.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

[illegible]

Russell A. Meier, Major, USAF
Project Officer

TABLE OF CONTENTS

TABLE OF CONTENTS (Concl'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

GLOSSARY

The following terms and acronyms are used in a more or less specialized sense herein.

Block	a floating-boundary block
Checking mode	a state wherein all subscripts of stores, loads and swaps are checked for legality
CM	checking mode
Dimension word	a preface word which defines the upper subscripting limit for a given dimension
DW	dimension word
Dynamic allocation	assignment of memory at execution time as opposed to load time
EB	extended block
Extended block	a block together with its preface words
Floating-boundary block	a dynamically allocated unit of storage
Free block	a block in Scratchpad which is not reserved
FUDGE	a constant used in subscript calculation (preface words)
II	the Ith subscript in a store, load or swap request
Key word	a word within the user's program which is associated with a block and is used to identify it
KW	key word
NDIM	the number of dimensions proper to a given block (Storage Control Word)
NI	the highest value that a subscript may legally assume in the Ith direction
NSXPAD	the total number of cells in Scratchpad

Name word	a word containing the name and data type of a block (preface words)
NW	name word
Packing	collecting of all reserved blocks together so as to create a single free block of maximum size
Preface words	cells just preceding a block intended for XPSTC's internal use (viz: SCW, NW, FUDGE (if any), and DW's (if any))
PWs	preface words
Reserved block	an allocated block
Scratchpad	the section of memory set aside for all dynamically allocated storage
SCW	storage control word
SM	stationary mode
Stationary mode	a state wherein packing is illegal
Storage control word	a word defining: (1) the associated KW; (2) the allocation status; (3) whether or not another block follows; (4) NDIM; and (5) the block size for a given block
XPAD	scratchpad
XPSTC	scratchpad storage controller

SECTION I

INTRODUCTION

PURPOSE AND SCOPE

XPSTC, a FORTRAN dynamic storage allocator, was originally designed to keep track of the floating-boundary storage blocks in a MITRE Project 4164 (SLBM) program. This program required floating-boundary storage because both input parameters and intermediate results could cause certain tables to vary in required size. To have fixed all such tables at their maximum size would have caused available core space to be exceeded. Furthermore, in order to assure that space was used only for live data (data with potential future use), it was desirable that programs share working storage space. XPSTC was written in a generalized fashion so as to be useful whenever situations of this sort arise. XPSTC performs four basic functions:

(1) allocation of storage blocks with dimensions as required by the calling program;

(2) in-flight adjustment of dimensions as may be required;

(3) storing to and loading from such blocks; and

(4) release of any storage which contains information which is no longer of use.

In addition, the program has six principal auxiliary functions:

(1) Checking Mode: When desired, all subscripts are monitored to assure that original dimensions are not exceeded during storing and loading.

(2) Infinite Storage Request: All available memory may be allocated when necessary.

(3) Direct Access: The calling program may obtain the absolute location of a given block, and perform its own storing and loading.

(4) Stationary Mode: As an adjunct to direct access, the status of Scratchpad may be temporarily frozen so that absolute addresses do not become obsolete because of block shuffling.

(5) Status Request: The calling program may obtain the status (reserved or free) of a given block.

(6) Debugging Aids: The calling program may request that a map or a complete dump of Scratchpad storage area be printed output. Also, all input (except subscripts when not in the checking mode) are monitored for validity.

GENERAL DESCRIPTION

XPSTC is a group of subroutines which may be called by the FORTRAN (or STRAP) user when he wishes one of the above functions carried out. It is totally dependent upon information given to it via such calls. It must be told, for example, how much memory is available for allocation and where this memory is. It will neither allocate, adjust, nor release a storage block without an explicit call.

In consequence of this, the first call must be to the initialization subroutine to inform XPSTC where Scratchpad (see Glossary) is and how much memory is available there. Storage is allocated from Scratchpad when a call is made to another subroutine, giving it a key word which will be the block identifier, and the dimensions of the required block. Any number of dimensions are allowable, including Ø which is interpreted as a request for a single cell. Another subroutine will allocate, as a one-dimensional (vector) block, all of Scratchpad which is currently free and inform the calling program as to the size of this block.

"Allocation" of a block consists of:

(1) storing a code in the key word supplied by the user, so that the intended block may be identified on subsequent store, load, or release requests; and

(2) storing of preface words which precede the reserved block in Scratchpad, delimit it, and flag it as reserved.

If sometime after a block has been allocated, it becomes necessary to alter the dimension specifications while preserving some or all of the data already stored into the block, it is possible to change any of the dimensions (though not the number of dimensions) by calling a subroutine designed for this purpose. Any element is preserved whose subscripts are legal under both sets of dimension specifications.

Storing into the allocated block is also accomplished by a subroutine call, giving the value to be stored, the key word, and the subscripts of the element. Any type of data (floating point, fixed point, logical, binary, or alphanumeric) may be stored. As is explained more fully in Section II, the "calling sequence," which is compiled from the FORTRAN user's CALL statement, is often replaced during execution by an in-line code which eliminates the time-consuming linkage process.

Loading from the block takes the form of a function-type reference, again giving the key word of the block and the subscripts of the element to be loaded.

A "swap" subroutine is also available; the parameters are similar to those for storing except that the value to be stored is itself replaced by the old value of the element in the storage block. As with the store subroutines, the calling sequences to load or swap are usually replaced by equivalent in-line code for the sake of speed.

When through using a block, the user may make the space therein available for other purposes by "releasing" the block. This is done by calling a subroutine, giving it the key word of the block to be released. Releasing the block consists of:

(a) disabling the key word so that attempted use (stores, loads, or swaps) of the block before another allocation request becomes illegal; and

(b) flagging of the block itself as "free."

In special circumstances it may be necessary to know the absolute address of an allocated block; for example, when it is desired to include a vector within the block among the call parameters to a general subroutine. In this situation it is possible to call a subroutine which, given the key word of the block, will return the "base address" of the block with respect to the first address of Scratchpad. In FORTRAN parlance, the value returned is the index of the cell just preceding the block, when all of Scratchpad is considered a vector array. The restrictions applying to this procedure, however, should be noted carefully (see Section II, H, page 22).

The other auxiliary functions mentioned in Section I are also carried out upon special subroutine call.

SECTION II

USAGE

Program input-output is achieved via calling sequence. In the case of an error, a diagnostic printout is supplied. A summary of calling sequences is shown in Table I.

CALLING SEQUENCES

The following is a detailed description of the individual calling sequences.

A. INSTO - Initialize Storage

CALL INSTO (XPAD, NSXPAD, CM)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
XPAD	Variable Array	---	Entire portion of memory which is to be used for floating boundary storage allocation.
NSXPAD	Fixed Point Expression	≥ 3	Number of memory cells in Scratchpad.
CM	Logical Expression	---	Logical .TRUE. if checking of subscripts during storing and loading is desired; .FALSE. otherwise.

(5) Operation (see Figure 1)

(A) If INSTO has been called previously in this computer run, all blocks allocated since the last such call are released.

(B) The address and size of Scratchpad are stored internally for use by the other XPSTC subroutines.

TABLE I

Call Sequence Summary Alphabetical, by Call Name

Function	Sequence	Paragraph
Adjust Dimension	CALL ADJD(KW, I, NI)	II. F
Allocation Status Test	XL =ALLOC (KW)	II. M
Req. All Avail. Storage	CALL ALSTO(KW, N)	II. E
Enter Checking Mode	CALL ENTCM	II. N
Enter Stationary Mode	CALL ENTSM	II. P
Floating Load	XX =FL(KW, (I1, . . . , INDIM)	II. J
Req. Block Base Index	IB =IBASE(KW)	II. H
Integer Load	II =IL(KW, I1, . . , INDIM)	II. J
Initialize Storage	CALL INSTO(XPAD, NSXPAD, CM)	II. A
Logical Load	XL =LL(KW, I1, . . , INDIM)	II. J
Resume Prior CM	CALL RESCM	II. O
Resume Prior SM	CALL RESSM	II. Q
Reassign Key Word	CALL RKW(OKW, NKW)	II. R
Release Storage	CALL RLSTO(KW1, KW2, . . .)	II. L
Request Named Storage	CALL RQNSTO(SYMBOL, KW, N1, . . . , NNDIM)	II. C
Request Storage	CALL RQSTO(KW, N1, . . . ,NNDIM)	II. B
Request XPAD Dump	CALL RQXDP	II. T
Request XPAD Map	CALL RQXMP	II. S
Store	CALL ST(X, KW, I1, . . , INDIM)	II. I
Swap	CALL SW (X, KW, I1, . . , INDIM)	II. K
Trial Adjust Dimension	XL =TADJD(KW, I, NI)	II. G
Trial Request Storage	XL =TRQSTO(KW, N1, . . , NNDIM)	II. D

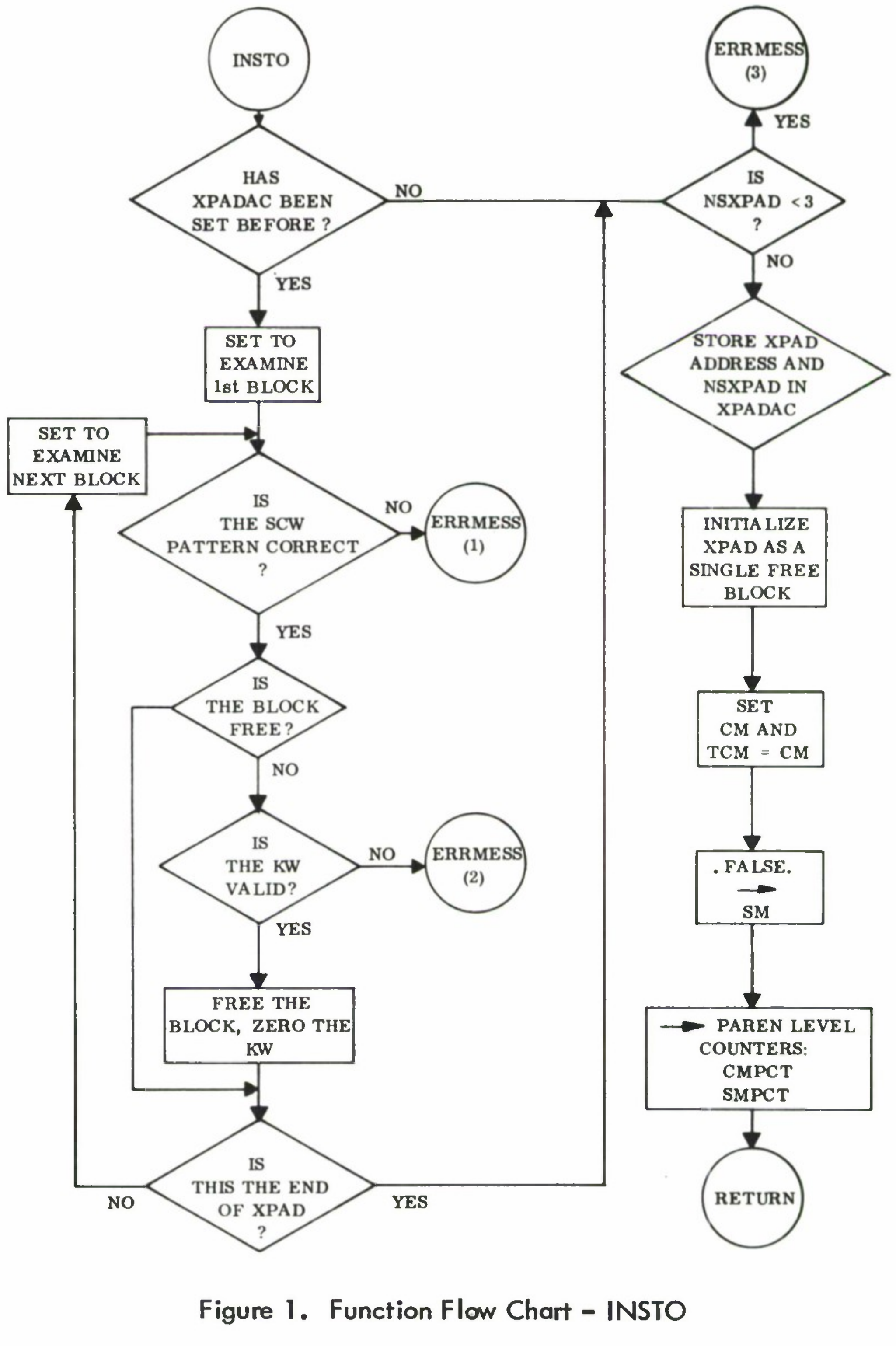

INSTO
HAS XPADAC BEEN SET BEFORE ?
NO
YES
SET TO EXAMINE 1st BLOCK
SET TO EXAMINE NEXT BLOCK
IS THE SCW PATTERN CORRECT ?
NO
ERRMESS (1)
YES
IS THE BLOCK FREE ?
NO
IS THE KW VALID?
NO
ERRMESS (2)
YES
FREE THE BLOCK, ZERO THE KW
IS THIS THE END OF XPAD ?
NO
YES
ERRMESS (3)
YES
IS NSXPAD < 3 ?
NO
STORE XPAD ADDRESS AND NSXPAD IN XPADAC
INITIALIZE XPAD AS A SINGLE FREE BLOCK
SET CM AND TCM = CM
.FALSE. → SM
→ PAREN LEVEL COUNTERS: CMPCT SMPCT
RETURN

Figure 1. Function Flow Chart – INSTO

(C) Checking mode is turned on or off as CM is .TRUE. or .FALSE., except that if checking mode has previously been off during the run, any stores, loads, or swaps executed while CM was off effectively, remain in the nonchecking mode.

(D) Stationary mode is turned off.

(E) Level counters [see Paragraphs N(2) and P(2)] for CM and SM are set to Ø.

(6) Comments and Examples

(A) It is frequently convenient to make XPAD synonymous with blank common:

```
      COMMON    XPAD    (2ØØØØ)
        .
        .
        .
      CALL      INSTO   (XPAD, 2ØØØØ, .FALSE.)
```

In this way any subroutine will have access to XPAD itself as an array variable; this is a prerequisite when using the direct access feature (see Paragraph H).

(B) Although it is more usual to leave Scratchpad itself fixed during a run, calling INSTO after the first time only when necessary to free all active blocks, there is no requirement that XPAD and NSXPAD do not change from call to call.

(C) The substance of (5) (C) above is that it is unusual, though not illegal, to change from nonchecking mode to checking mode via INSTO during the course of a computer run. In the usual case, CM is an external input parameter which is set to true at the beginning of runs in early checkout phases of program development and then is set to false. Storing, loading, and swapping are much faster in the nonchecking mode.

B. RQSTO - Request Storage

CALL RQSTO (KW, N1, N2 ,. . . , N1 ,. . . , NNDIM)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
KW	Variable	May not be within XPAD	The key word which will be used to identify the resulting block.
NI	Fixed Point Expression	>ø	Upper subscripting limit for the Ith dimension of the requested block; if dimension is wholly absent, one cell is allocated.

(5) Operation (see Figures 2 and 3)

(A) The size of the required extended block is determined by the following formulae:

NDIM	Extended Block Size
ø	3
1	N1 + 2
2	(N1)(N2) + 4
$\geq$ 3	(N1) (N2). . (NNDIM) + NDIM + 3

(B) Scratchpad is searched for a free block large enough to accommodate this extended block. If one is found, the sequence continues with (D) below.

(C) Otherwise, Scratchpad is packed (if in SM, an error stop occurs). If the free block thus created is still not large enough, an error stop results.

(D) The address of this extended block is stored into the key word. The user should not alter the key word, at least until the allocated storage has been released [1] or the key word has been reassigned [see Paragraph R].

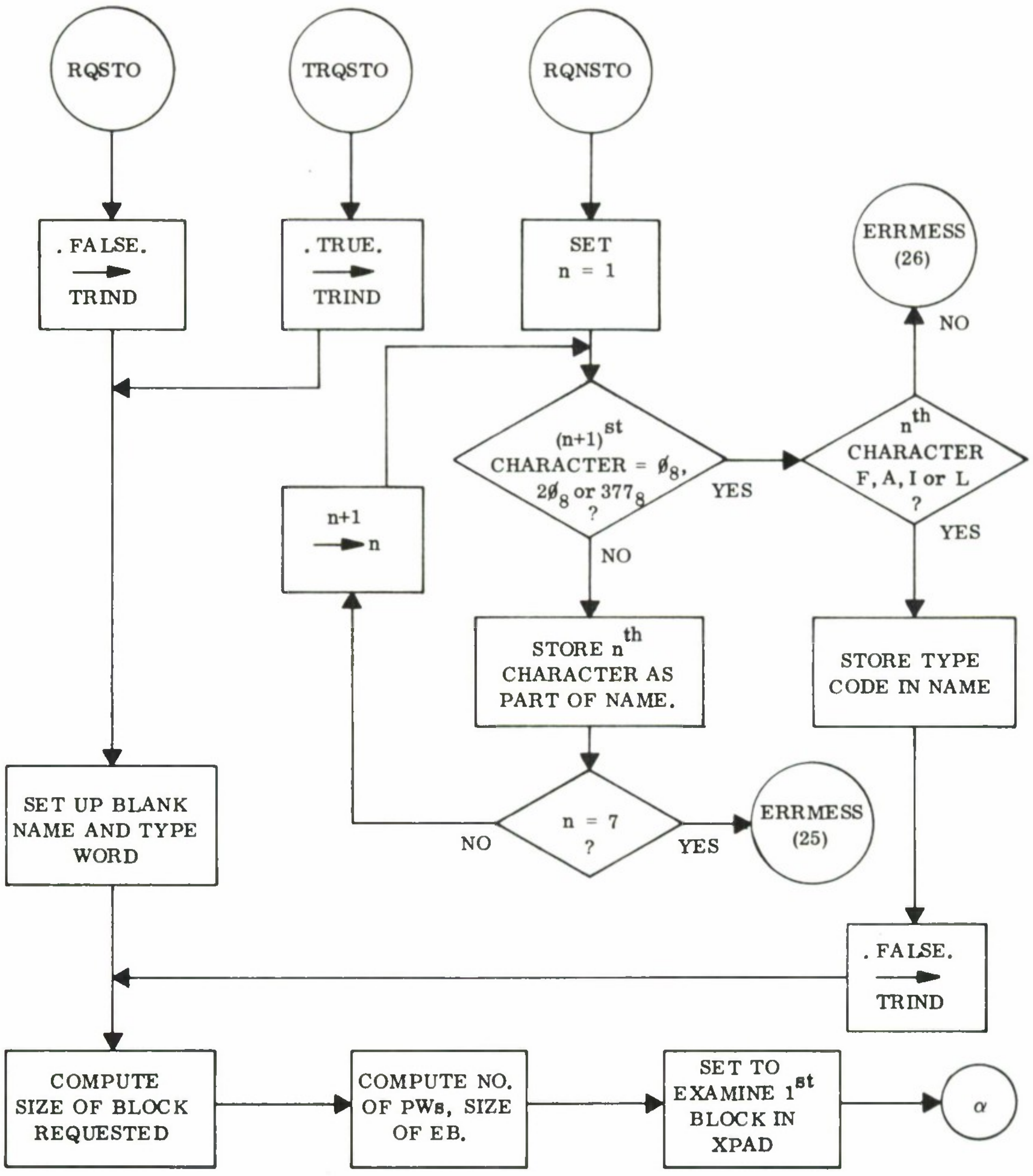

RQSTO
TRQSTO
RQNSTO
.FALSE. → TRIND
.TRUE. → TRIND
SET n = 1
ERRMESS (26)
NO
(n+1)st CHARACTER = Ø8, 2Ø8 or 3778 ?
YES
nth CHARACTER F, A, I or L ?
YES
n+1 → n
NO
STORE nth CHARACTER AS PART OF NAME.
STORE TYPE CODE IN NAME
SET UP BLANK NAME AND TYPE WORD
n = 7 ?
NO
YES
ERRMESS (25)
.FALSE. → TRIND
COMPUTE SIZE OF BLOCK REQUESTED
COMPUTE NO. OF PWs, SIZE OF EB.
SET TO EXAMINE 1st BLOCK IN XPAD
α

Figure 2. Functional Flow Chart - RQSTO, RQNSTO, TRQSTO

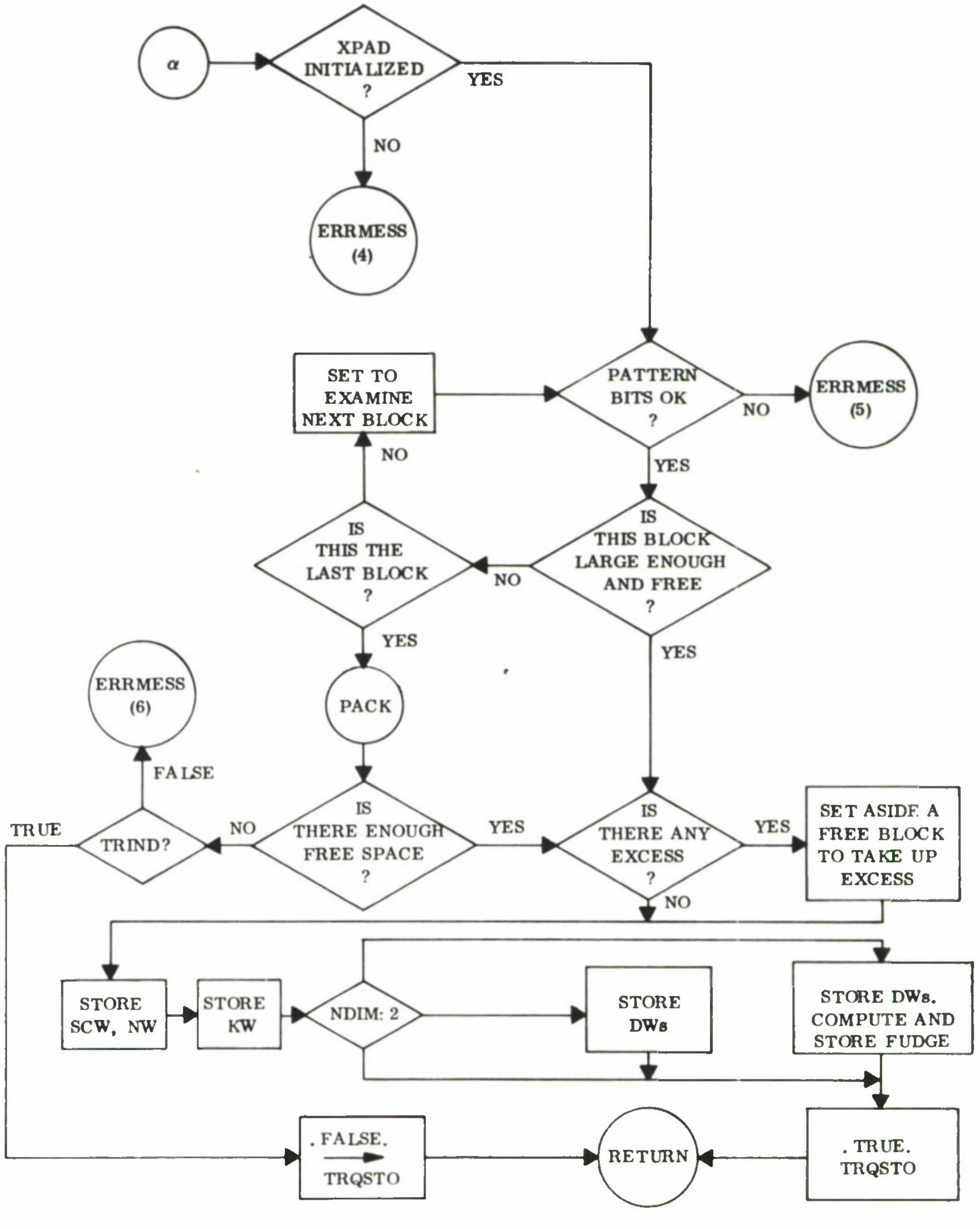


Figure 3. Functional Flow Chart - RQSTO, RQNSTO, TRQSTO

(E) The necessary information is placed in the preface words (the NW is zeroed), a free block is created to take up any left-over space, and the subroutine returns.

(6) Comments and Examples

(A) CALL RQSTO (KW), a Ø (dimension storage request) has been made legal for two reasons: symmetry, and such requests would be necessary if a data reader were added to XPSTC. With the present XPSTC, such a request has no rational application. The KW cell itself might as well be used for the storage, and the resulting access code generated by FORTRAN would be a great deal more efficient.

(B) In general, when the maximum attainable size of a table is known at compile-time to be quite small, 10 or 20 cells, it is not worthwhile to use floating memory. An ordinary DIMENSION statement requires less programming and less trouble, saves a little time, and may waste no more storage in the long run.

(C) The sequence:

```
CALL RQSTO      (X, NOOFXS)
CALL RQSTO      (Y, 16, NOOFYS+NNS, NGS, NFS)
```

would cause two blocks to be reserved: one would be a 1-dimensional array with NOOFXS storage cells and two preface words; the other would be a 4-dimensional array with (16) (NOOFYS+NNS) (NGS) (NFS) data cells and seven preface words.

C. RQNSTO - Request Named Storage

CALL RQNSTO (SYMBOL, KW, N1, N2, . . , N1, . . . , NNDIM)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
SYMBOL	Variable	Contains A8 code	The name to be assigned the block (up to six characters) plus a type character (F, floating; A, alphanumeric; I, integer; L, logical) for storage.

C. RQNSTO - Request Named Storage (Cont.)

CALL RQNSTO (SYMBOL, KW, N1, N2, . . , N1, . . . , NNDIM),

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
KW	Variable	Not in XPAD	The key word which will be used to identify the resulting block.
NI	Fixed Point Expression	>ø	Upper subscripting limit for the I^th dimension of the requested block; if dimensioning is wholly absent, one cell is allocated.

(5) Operation (see Figures 2 and 3)

Except that the block name and type are extracted from SYMBOL and stored in the NW, operation is exactly that of RQSTO [see Paragraph B].

(6) Comments and Examples

(A) In the current XPSTC, RQNSTO adds nothing to RQSTO, except perhaps readability to the output from RQXMP [see Paragraph S].

(B) The call

CALL RQNSTO (3HXAF, X, NOOFXS) causes a one-dimensional vector with NOOFXS storage cells and two preface words to be reserved. The name of the block is considered to be XA and the type of data is considered floating. Note, however, that data, going in and out of a storage block via stores, loads or swaps, is not monitored for type whether the block is named or not.

D. TRQSTO - Trial Request Storage

Logical Function: TRQSTO(KW, N1, N2, . . . , N1, . . . NNDIM)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
KW	Variable	May not be within XPAD	Key word which will be used to identify the resulting block, if any.

Logical Function: TRQSTO (KW, N1, N2, . . . , NI, . . . , NNDIM), (Cont.)

(1) Parameter	(2) Type	(3) Restriction	(4) Description
NI	Fixed Point Expression	>ø	Upper subscripting limit for the I^{th} dimension of the requested block; if dimensioning is wholly absent, one cell is implied.

(5) Operation (see Figures 2 and 3)

Operation is exactly that of RQSTO except that if sufficient storage for the requested block is not available, the job is not aborted but rather the function value is set to . FALSE. and no allocation is made. When a normal allocation occurs, the function takes on the value .TRUE.

(6) Example

The IF statement in the sequence (non-SM)

```
          LOGICAL TRQSTO
          IF (TRQSTO (SAVE, NRES, 5ø)) GO TO 3øø
     C    NOT ENOUGH ROOM TO SAVE INTERMEDIATE
     C    RESULTS - MUST USE DISK
      2øø----------------
     C    SAVE AREA ALLOCATED, SAVE RESULTS
      3øø----------------
```

will cause allocation of SAVE and a branch to statement 3øø if NRES*50+4 locations are free; otherwise the sequence starting at statement 2øø will be executed.

E. ALSTO - Request All Available Storage

CALL ALSTO (KW, N1)

(1) Parameter	(2) Type	(3) Restriction	(4) Description
KW	Variable	Not in XPAD	Key word for the block.

CALL ALSTO (KW, N1), (Cont.)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
N1	Fixed Point Variable	---	Returned as the number of data cells reserved. N1 may be returned as Ø, which means that less than 3 cells were found to be free in Scratchpad and no block was reserved.

(5) Operation (see Figure 4)

(A) Scratchpad is packed unless in SM. In this case, the largest free block is found.

(B) If there are less than 3 cells in this block, return N1=Ø.

(C) Set up key word and two preface words. Set N1=number of available cells minus two and return. This completes the set up of one-dimensional vector of N1 data words.

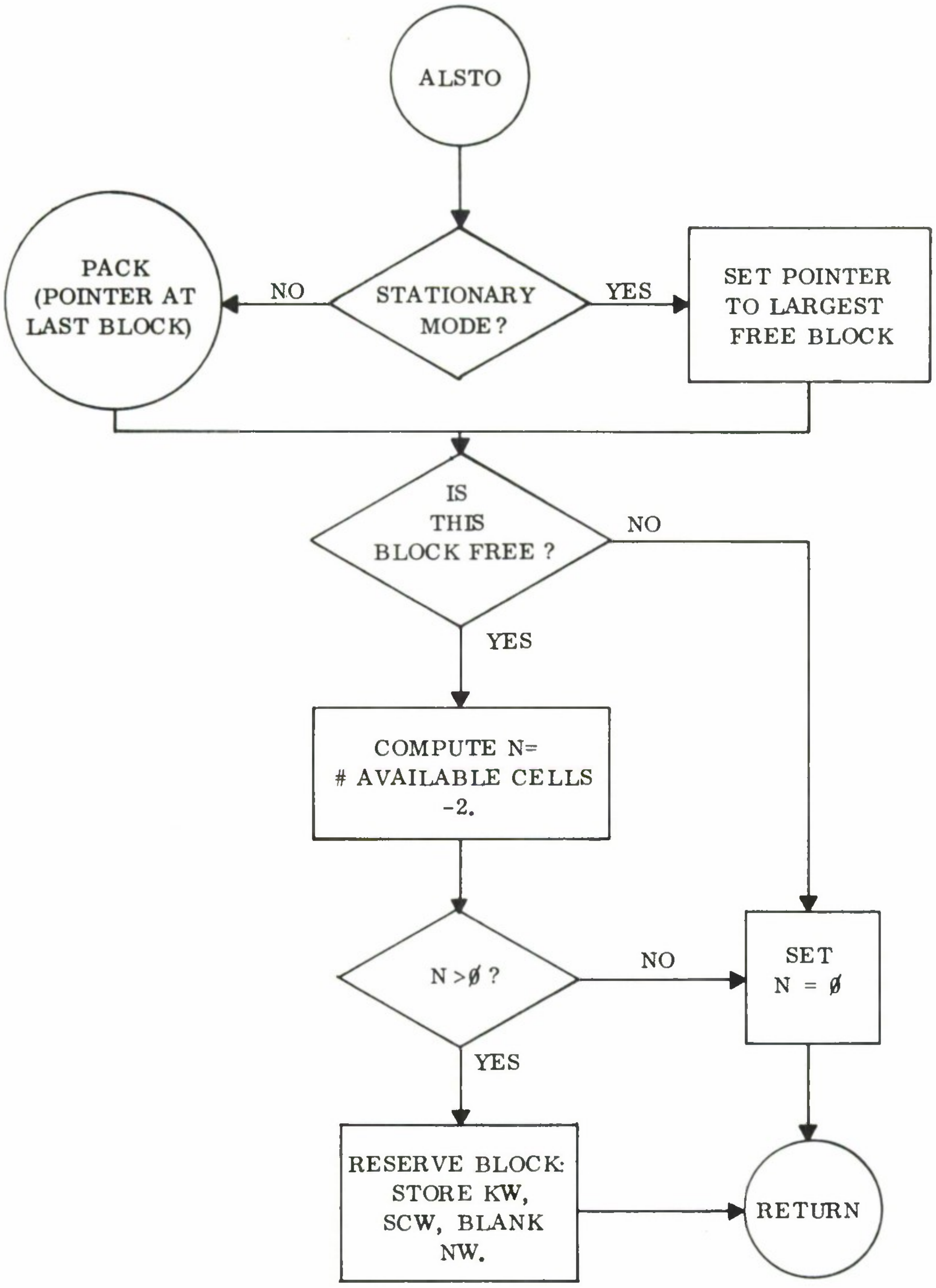
(6) Comments and Examples

The call CALL ALSTO (X, NX) has the same result as if NX could be determined in advance and (NX>Ø) CALL RQSTO (X,NX) were executed, except for the physical order of the blocks within Scratchpad.

F. ADJD - Adjust Dimension

CALL ADJD (KW, I, NI)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
KW	Variable	See (6)(A) below	Key word of the block whose dimensions are to be adjusted.
I	Fixed Point Expression	1≤I≤NDIM	No. of the dimension whose subscripting limit is to be adjusted.



ALSTO
STATIONARY MODE?
NO
YES
PACK (POINTER AT LAST BLOCK)
SET POINTER TO LARGEST FREE BLOCK
IS THIS BLOCK FREE ?
NO
YES
COMPUTE N= # AVAILABLE CELLS -2.
N >Ø ?
NO
YES
SET N = Ø
RESERVE BLOCK: STORE KW, SCW, BLANK NW.
RETURN

Figure 4. Functional Flow Chart - ALSTO

CALL ADJD (KW, I, NI), (Cont.)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
NI	Fixed Point Expression	>ø	New upper subscripting limit for the I^th dimension.

(5) Operation (see Figures 5, 6, and 7)

(A) The block associated with KW is located. It must be currently allocated.

(B) If the new NI is equal to the old NI, an immediate return is made.

(C) If the new NI is less than the old NI, the internal data is rearranged to match the new dimensioning, a free block is created to take up the excess, the preface words are updated, and a normal return is made.

(D) If the new NI is greater than the old NI, operation depends upon the arrangement of Scratchpad. If there is sufficient space in the form of free blocks immediately above this block, the action taken is similar to (C) above. If, as is usually the case, this condition is not met, Scratchpad is packed so that all free storage is located after this block (in stationary mode) and an error stop results. If this is still not enough, an error stop results; otherwise, continue as in (C) above.

(6) Comments and Examples

(A) The key word used must be exactly that used when the block was originally reserved, or a derivative via RKW [see Paragraph R]. The sequence

```
CALL RQSTO (A,NA1,NA2)
   .
   .
   .
B=A
   .
   .
   .
CALL ADJD (B,1,NA1+1)
```

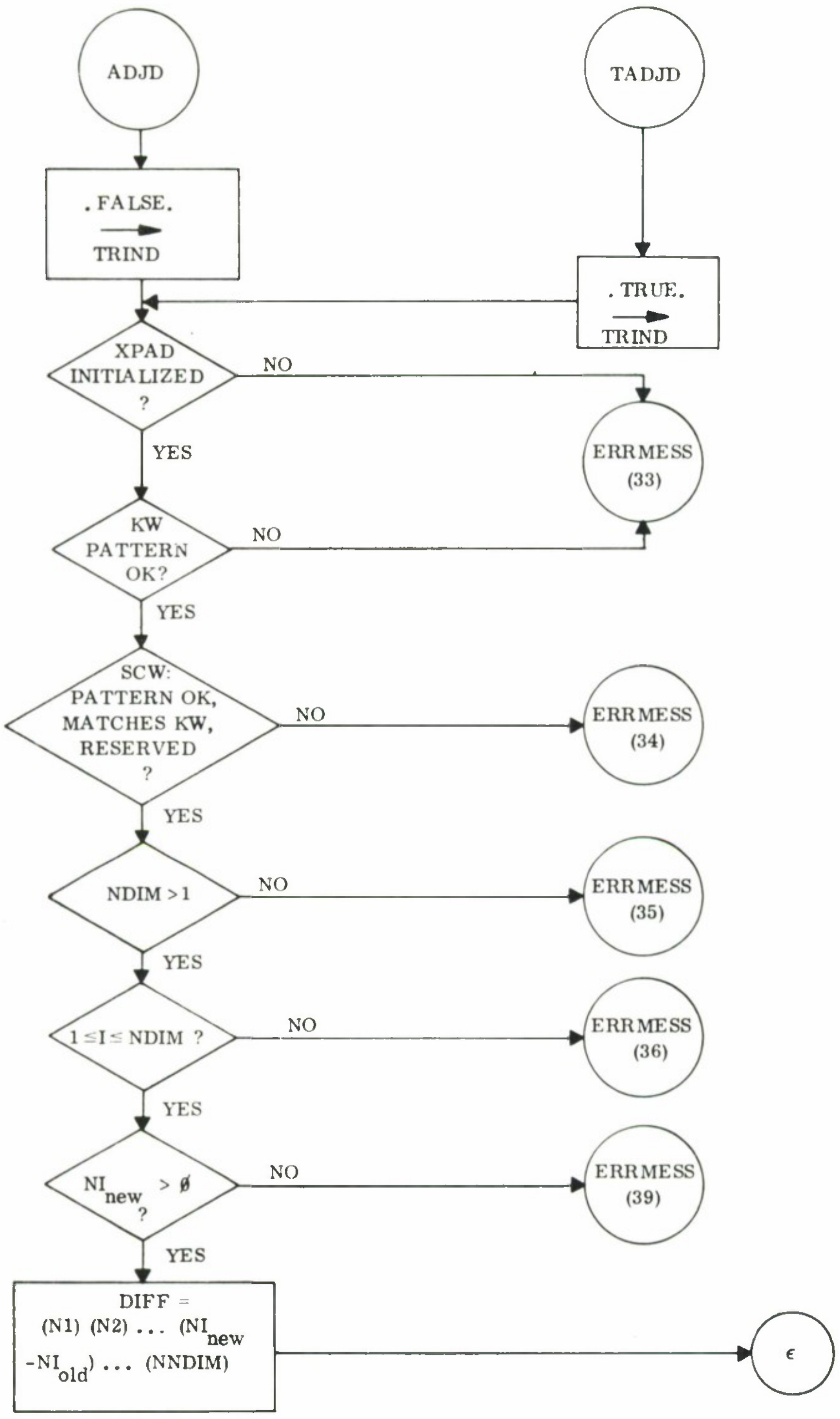


Figure 5. Functional Flow Chart - ADJD, TADJD

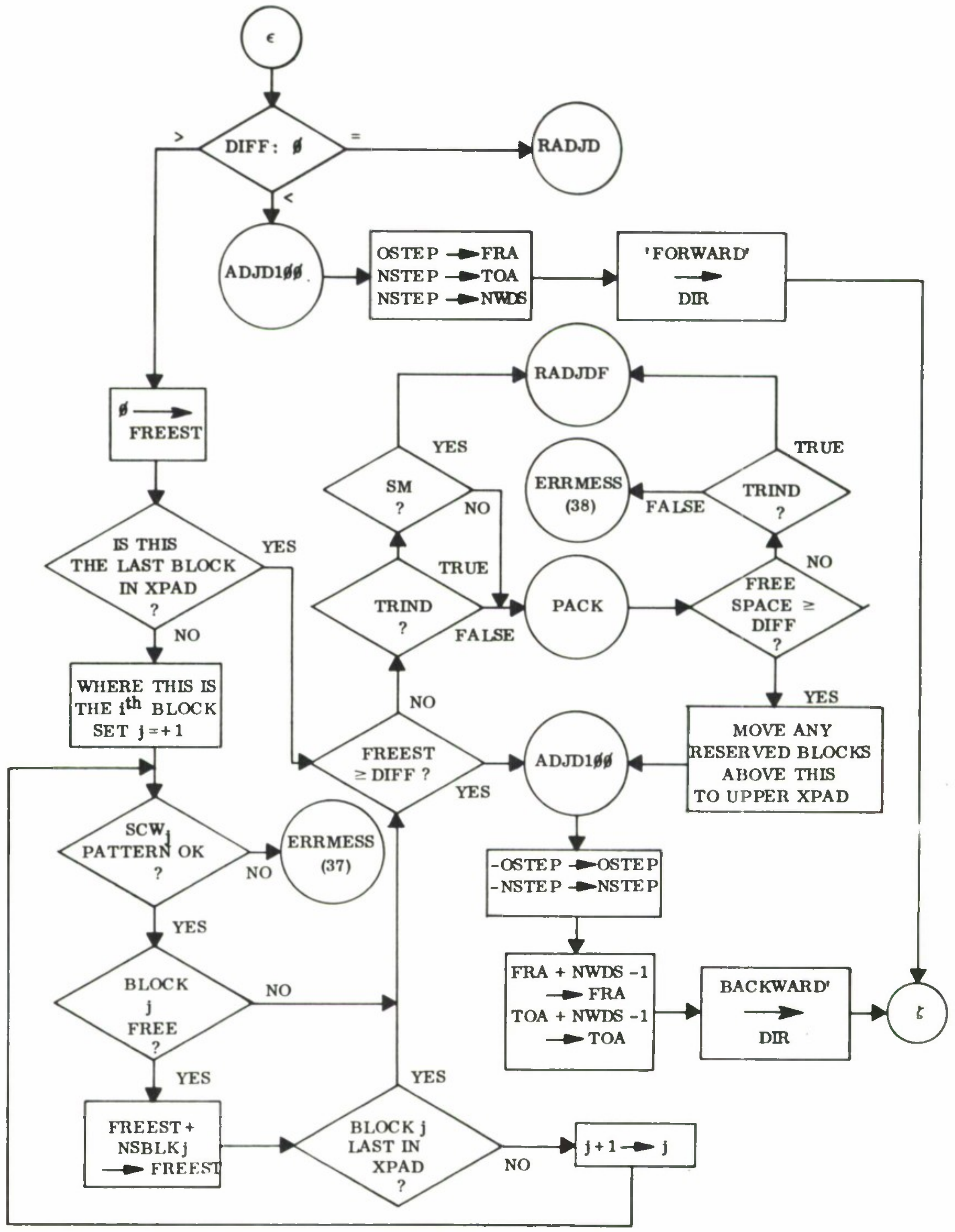


Figure 6. Functional Flow Chart - ADJD, TADJD

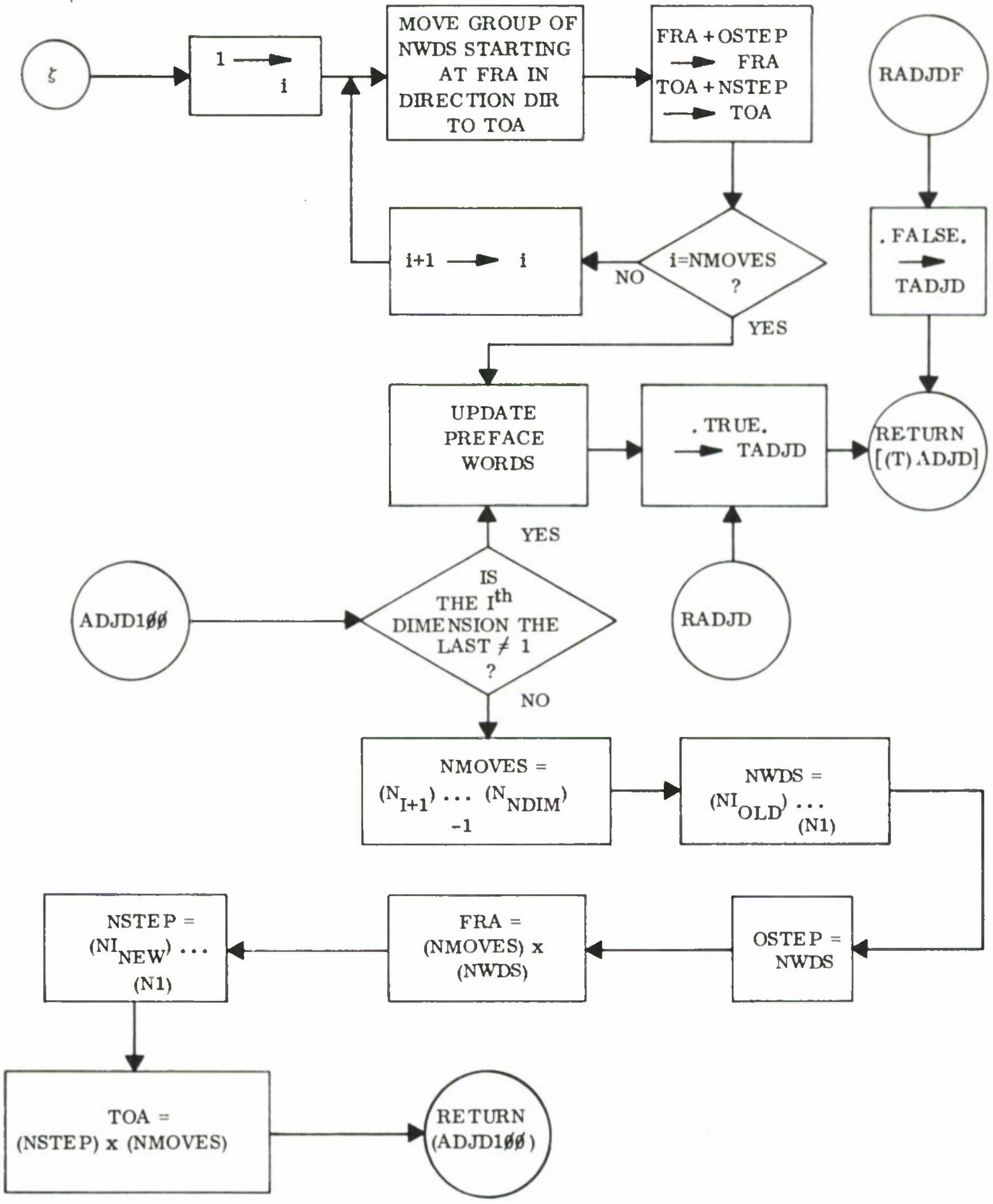


Figure 7. Functional Flow Chart – ADJD, TADJD

would result in an error stop. The sequence would be acceptable if B=A were replaced by CALL RKW (A, B). One consequence of this is that key words cannot ordinarily be passed through calling sequences to FORTRAN subroutines [1]* without resorting to oblique tricks. When more than one routine must access a block, the best technique is to place the key word for that block in some COMMON.

(B) The restriction on (1) Parameter I, implies NDIM≥1; an attempt to adjust the dimensions of a ∅-dimension block is always an error.

(C) When the new NI is less than the old NI, all elements whose I th subscript is greater than the new NI are irrevocably lost. In the reverse case, any elements whose I th subscript is greater than the old NI are in an uninitialized state and may have any value prior to being stored into.

(D) For all practical purposes, an ADJD with the new NI greater than the old should be considered illegal in stationary mode.

G. TADJD - Trial Adjust Dimension

Logical Function = TADJD (KW, I, NI)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
KW	Variable	See F.(6)(A)	Key word of the block whose dimensions are to be adjusted.
I	Fixed Point Expression	1≤I≤NDIM	No. of the dimension whose subscripting limit is to be adjusted.
NI	Fixed Point Expression	>∅	New upper subscripting limit for the I th dimension.

(5) Operation (see Figures 5, 6, and 7)

Operation is exactly that of ADJD, except that neither of the error stops mentioned in Paragraph F.(5)(D) can occur. Instead, no adjustment

*See Ref. 1; II.D.2 .10.1

is made and the function is assigned the value .FALSE.. When adjustment occurs normally, the function becomes .TRUE..

(6) Comments and Examples

(A) All comments for ADJD [see Paragraph F.(6)] apply except (D).

(B) Assuming sufficient space, the sequence

```
              LOGICAL TADJD
                .
                .
              CALL RQSTO (A,2,4,6,8)
                .
                .
                .
                .
              IF (.NOT. TADJD (A,3,9) GO TO 3ØØ
      2ØØ       .
                .
                .
                .
                .
```

would fall through to statement 2ØØ with block "A" appearing exactly as though it had been reserved by a

```
              CALL RQSTO (A,2,4,9,8)
```

H. IBASE - Request Block Base Index

Function: IBASE (KW)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
KW	Variable	See F.(6)(A)	Key word of the block whose base index is requested.

(5) Operation (see Figure 8)

(A) The block associated with KW is located. It must be currently allocated.

(B) The base index is computed so that if XPAD(1) denotes the first element in Scratchpad, XPAD (IBASE(KW)+1) denotes the first

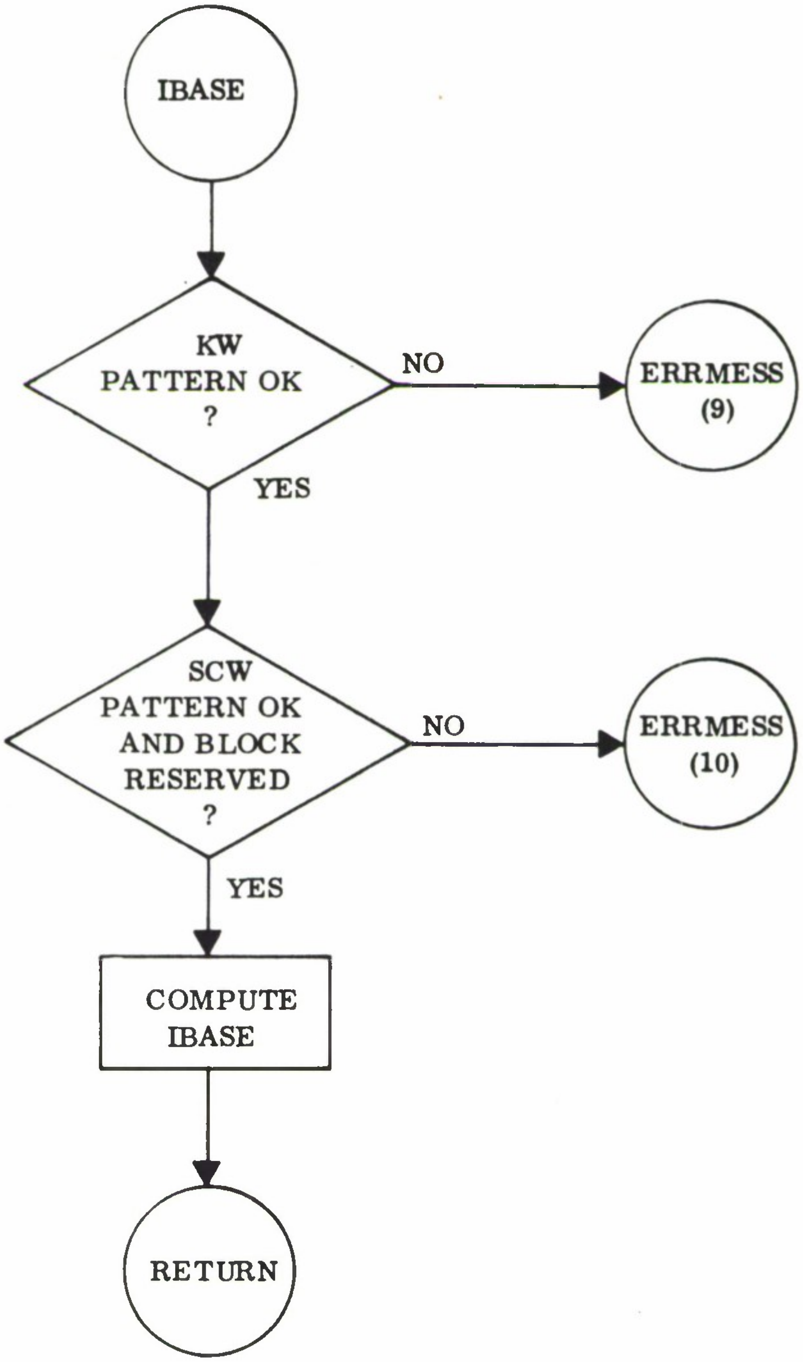


Figure 8. Functional Flow Chart – IBASE

element of the data storage, as opposed to preface words, associated with KW.

(6) Comments and Examples

The base index has only temporary validity, and caution should be exercised in using it. Whenever a storage-using call that is TRQSTO, RQNSTO,RQSTO, expanding TADJD or ADJD, ALSTO is made, the location of a particular block already in Scratchpad may be altered, unless stationary mode is in effect. The key word is updated to reflect the change, but there is no way for XPSTC to update indexes calculated from IBASE. Consequently the base index and the results of any concomitant index computations should not be used without recalculation after any storage-using call except in SM. The following situation represents a violation of this rule. Program II is a subroutine which, rather inefficiently, reverses the order of a vector X of NX entries. Program I wishes this operation performed upon the J th column of a 2-dimensional matrix. Note: FORTRAN, XPSTC stores with the first subscript varying most rapidly, column-wise, in the case of 2 dimensions. The matrix is in Scratchpad and the programmer of Program I forgets or does not know that Program II calls for storage before using the vector address:

PROGRAM I
(Assume non-SM)

```
COMMON XPAD (2ØØØØ)
  .
  .
CALL INSTO (XPAD, 2ØØØØ, .FALSE.)
  .
  .
CALL RQSTO (XMATR,N,M)
  .
  .
IXCOLJ = IBASE (XMATR)+N*(J-1)+1
CALL PROG II (XPAD(IXCOLJ),N)
  .
  .
  .
```

PROGRAM II

```
      SUBROUTINE PROG II(X,NX)
      DIMENSION X (NX)
      COMMON XPAD (2ØØØØ)
      CALL RQSTO (TEMP,NX)
      ITEMP = IBASE (TEMP)
      DO 1Ø I=1,NX
      INDEX = ITEMP+NX-I+1
1Ø    XPAD(INDEX) = X(I)
      DO 2Ø I = 1,NX
      INDEX = ITEMP+I
2Ø    X(I) = XPAD (INDEX)
      CALL RLSTO (TEMP)          [see Paragraph L]
      RETURN
```

The above sequence could very well blow up if the CALL RQSTO (TEMP, NX) caused a packing of Scratchpad so that XPAD (IXCOLJ) no longer was the first element of column J. As it stands, Program II is a correct program only, subject to the restriction that either none of the call parameters shall be Scratchpad addresses or that SM shall be in effect. If Program II were recoded as shown below, then Program I would be correct as it stands:

```
      SUBROUTINE PROG II (X,NX)
      DIMENSION X (NX)
      MX = NX/2
      J = NX+1
      DO 1Ø I = 1, MX
      J = J - 1
      TEMP = X(J)
      X(J) = X(I)
1Ø    X(1) = TEMP
      RETURN
```

Another way of correcting the situation would be for the PROG II programmer to bracket the CALL RQSTO with the statements

```
      CALL ENTSM [ p ]
      CALL RESSM [ q ]
```

or the PROG I programmer could do the same with the CALL PROG II statement.

I. ST - Store

CALL ST (X,KW,I1,I2,...,II, ...,INDIM)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
X	Expression	---	The value to be stored.
KW	Variable	See F.(6)(A)	Key word of the block to be stored into.
II	Fixed Point Expression	$\emptyset < II \leq NI$	Ith subscript of the element to receive the value X. The numbers of subscripts must match the number of dimensions in the storage request.

(5) Operation (see Figures 9 and 10)

(A) The block associated with KW is located. It must be currently allocated.

(B) The number of subscripts is checked on the first execution only, within a computer run.

(C) If XPSTC is in the checking mode, a number of validity checks are made upon the key word and preface words, and all the subscripts are checked for valid range. With these tests passed, the effective address of the cell is compiled from the subscripts, the store is made, and the program returns.

(D) On the first execution after nonchecking mode has been established, all the checks described in (C) are performed; however, before returning, a block of code is usually generated and stored over the calling sequence. This code is designed to perform the floating-block store in much the same way that FORTRAN ordinarily compiles its stores to static blocks; only one or two extra machine instructions are required in most cases. The only exception to this rule is the situation where one of the call parameters is a variable with a variable subscript. In this case, only the branch within the calling

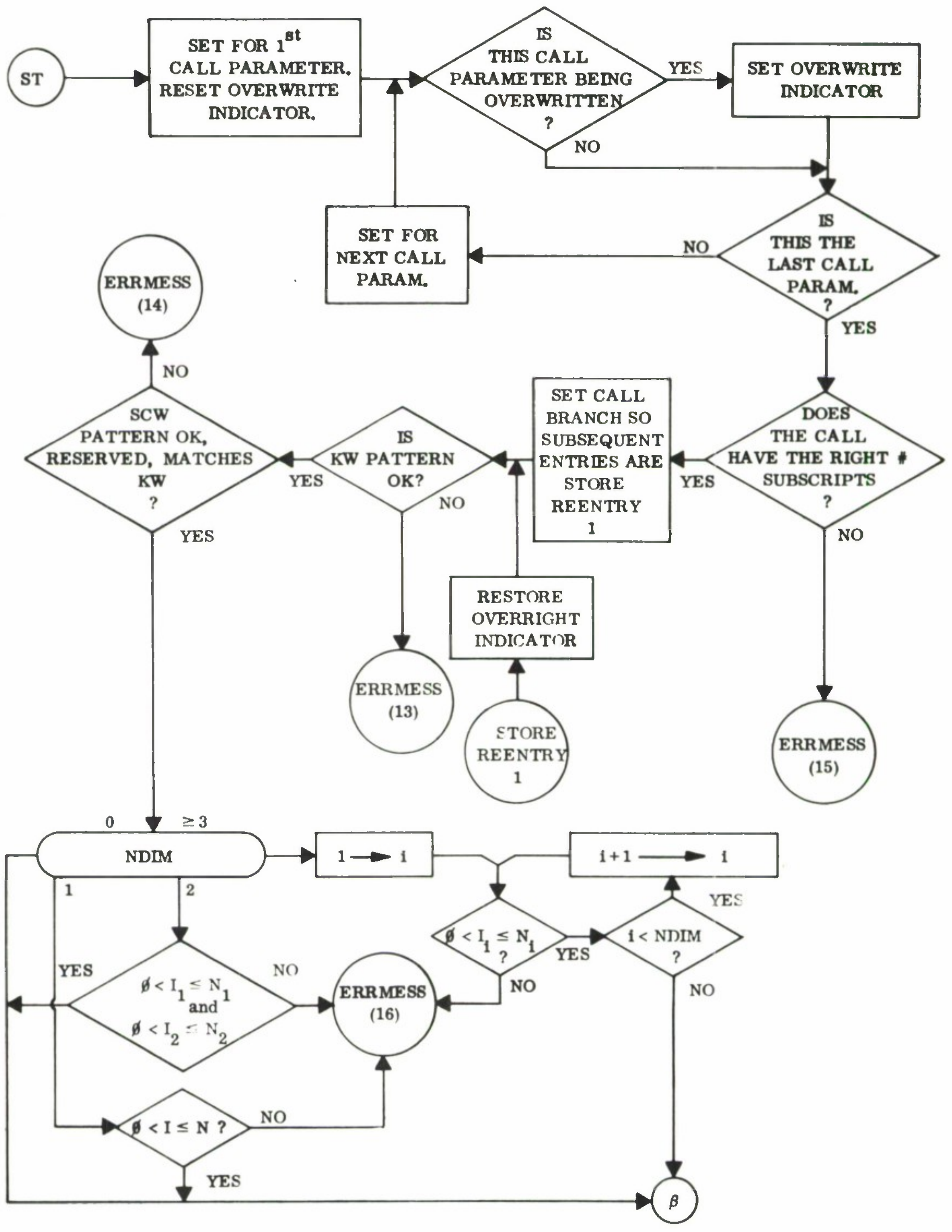
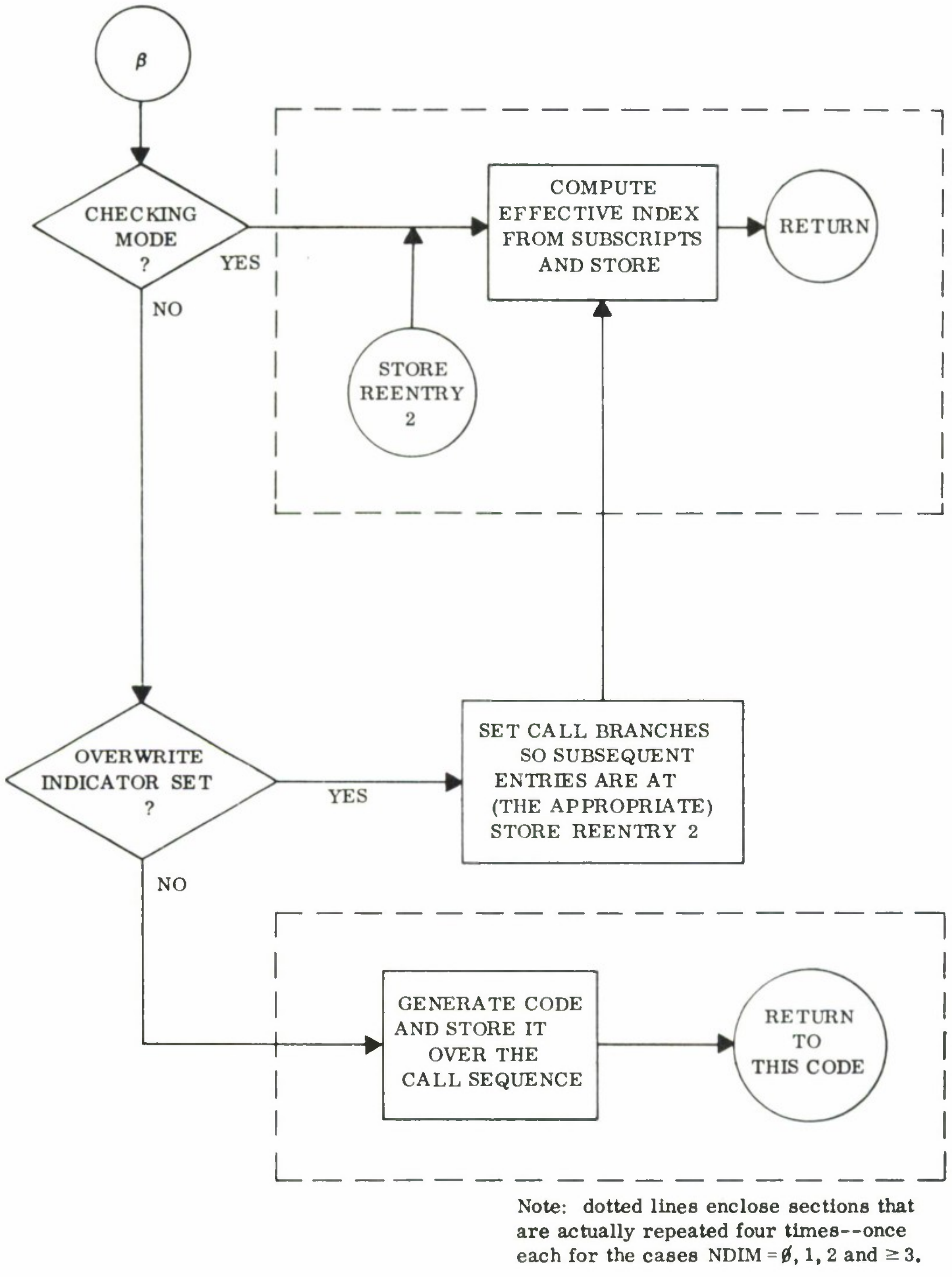

ST
SET FOR 1st CALL PARAMETER. RESET OVERWRITE INDICATOR.
IS THIS CALL PARAMETER BEING OVERWRITTEN ?
YES
NO
SET OVERWRITE INDICATOR
IS THIS THE LAST CALL PARAM. ?
NO
YES
SET FOR NEXT CALL PARAM.
ERRMESS (14)
NO
SCW PATTERN OK, RESERVED, MATCHES KW ?
YES
IS KW PATTERN OK?
YES
NO
SET CALL BRANCH SO SUBSEQUENT ENTRIES ARE STORE REENTRY 1
YES
DOES THE CALL HAVE THE RIGHT # SUBSCRIPTS ?
NO
ERRMESS (13)
RESTORE OVERRIGHT INDICATOR
STORE REENTRY 1
ERRMESS (15)
0
≥ 3
NDIM
1
2
1 → i
i+1 → i
YES
NO
ϕ < I1 ≤ N1 and ϕ < I2 ≤ N2
ERRMESS (16)
ϕ < Ii ≤ Ni ?
YES
NO
i < NDIM ?
YES
NO
ϕ < I ≤ N ?
NO
YES
β

Figure 9. Functional Flow Chart – ST


β
CHECKING MODE ?
YES
NO
COMPUTE EFFECTIVE INDEX FROM SUBSCRIPTS AND STORE
RETURN
STORE REENTRY 2
OVERWRITE INDICATOR SET ?
YES
SET CALL BRANCHES SO SUBSEQUENT ENTRIES ARE AT (THE APPROPRIATE) STORE REENTRY 2
NO
GENERATE CODE AND STORE IT OVER THE CALL SEQUENCE
RETURN TO THIS CODE

Note: dotted lines enclose sections that are actually repeated four times--once each for the cases NDIM = ∅, 1, 2 and ≥ 3.

Figure 10. Functional Flow Chart - ST

sequence is replaced so that a direct entry to a special fast store routine is made each call.

(6) Comments and Examples

(A) STRAP users may guess from (5)(D) above that the calling sequence to ST should not be stored into unless some signal is furnished XPSTC [see Section III, Storing, Loading and Swapping, page 58]. FORTRAN users especially concerned with efficiency should consult this section also.

(B) The sequence

```
        .
        .
        .
    CALL RQSTO (SINES, MAXROW, MAXCOL)
        .
        .
        .
    CALL ST (SIN (X)-1,SINES,MAX(I,J),K * N+L(I))
```

will result in the value sin x-1 being stored into the element of SINES which has the subscripts max (i, j), $kn+1_i$.

J. FL, IL, LL - Floating, Integer, and Logical Load

Functions: FL (KW, I1, I2, ... , II, . . . INDIM)
IL (KW, I1, I2, ... , II, . . . INDIM)
LL (KW, I1, I2, ... , II, . . . INDIM)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
KW	Variable	See F. (6)(A)	Key word for the block to be loaded from.
II	Fixed Point Expression	$\emptyset < II \leq INDIM$	I^th subscript of the element whose value will become the value of the function. The number of subscripts must match the number of dimensions in the storage request (NDIM).

(5) Operation (see Figures 11 and 12)

Operation is the same as that of ST [see Paragraph I], except that "store" should be read "load" and "stores to" should read "loads from".

(6) Comments and Examples

(A) All comments for ST [see Paragraph I] apply with obvious modifications.

(B) FL, IL, and LL are merely different names for exactly the same program. Since XPSTC neither knows nor cares what type of data the block is being used for, the effective value of any one of these functions will always be the sixty-four bit quantity within the subscripted cell. For STRAP users, this means that the accumulator and all associated indicators have been set by a LWF(U) just prior to return. The motivation for separate names is to allow the FORTRAN user, with a LOGICAL LL statement, to avoid mixed expressions and otherwise control the FORTRAN handling of the function value.

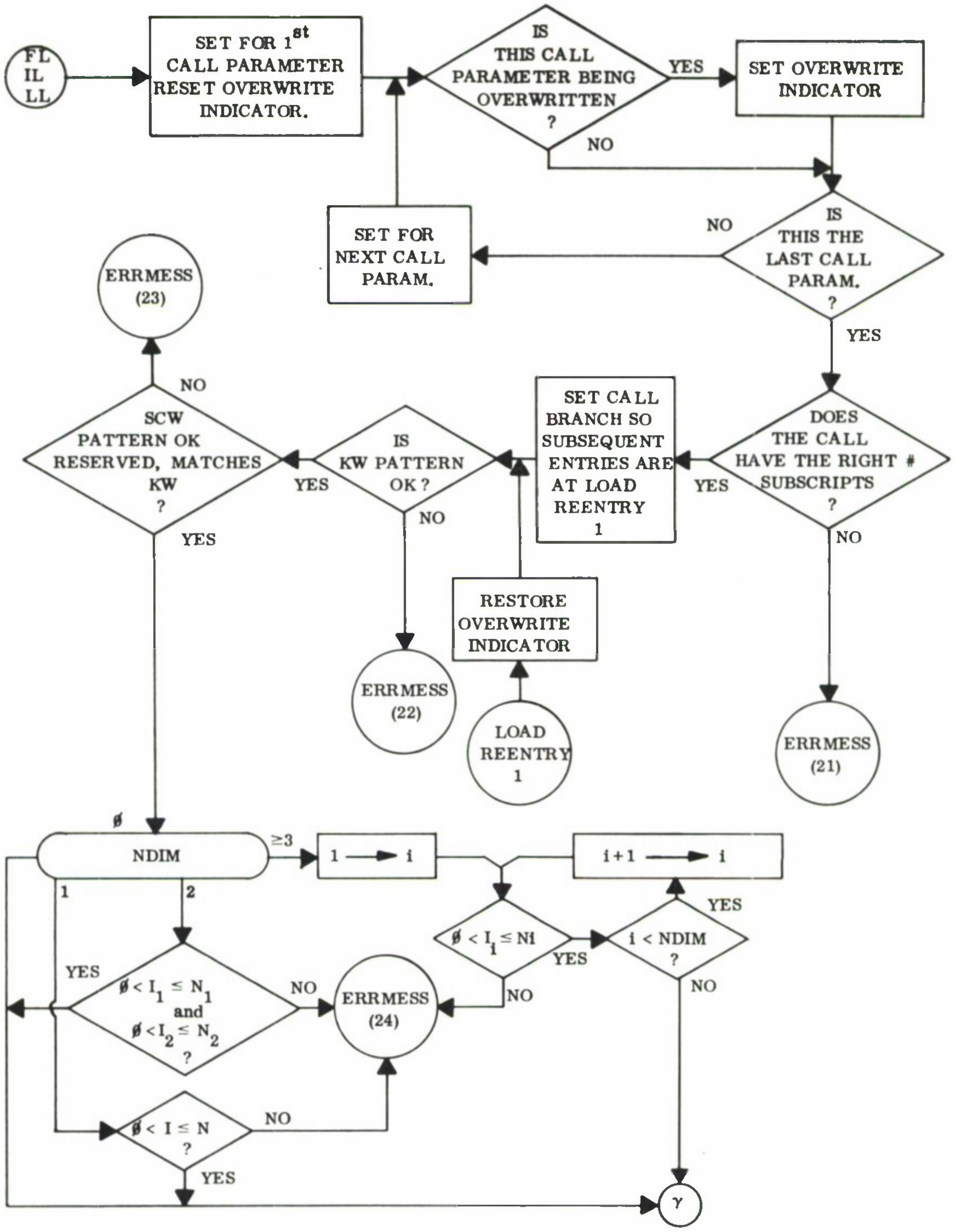
(C) The following statement will cause the I, Jth element of a block of counters to be incremented:

```
CALL ST (IL(KOUNT,I,J)+1,KOUNT,I,J)
```

K. SW - Swap

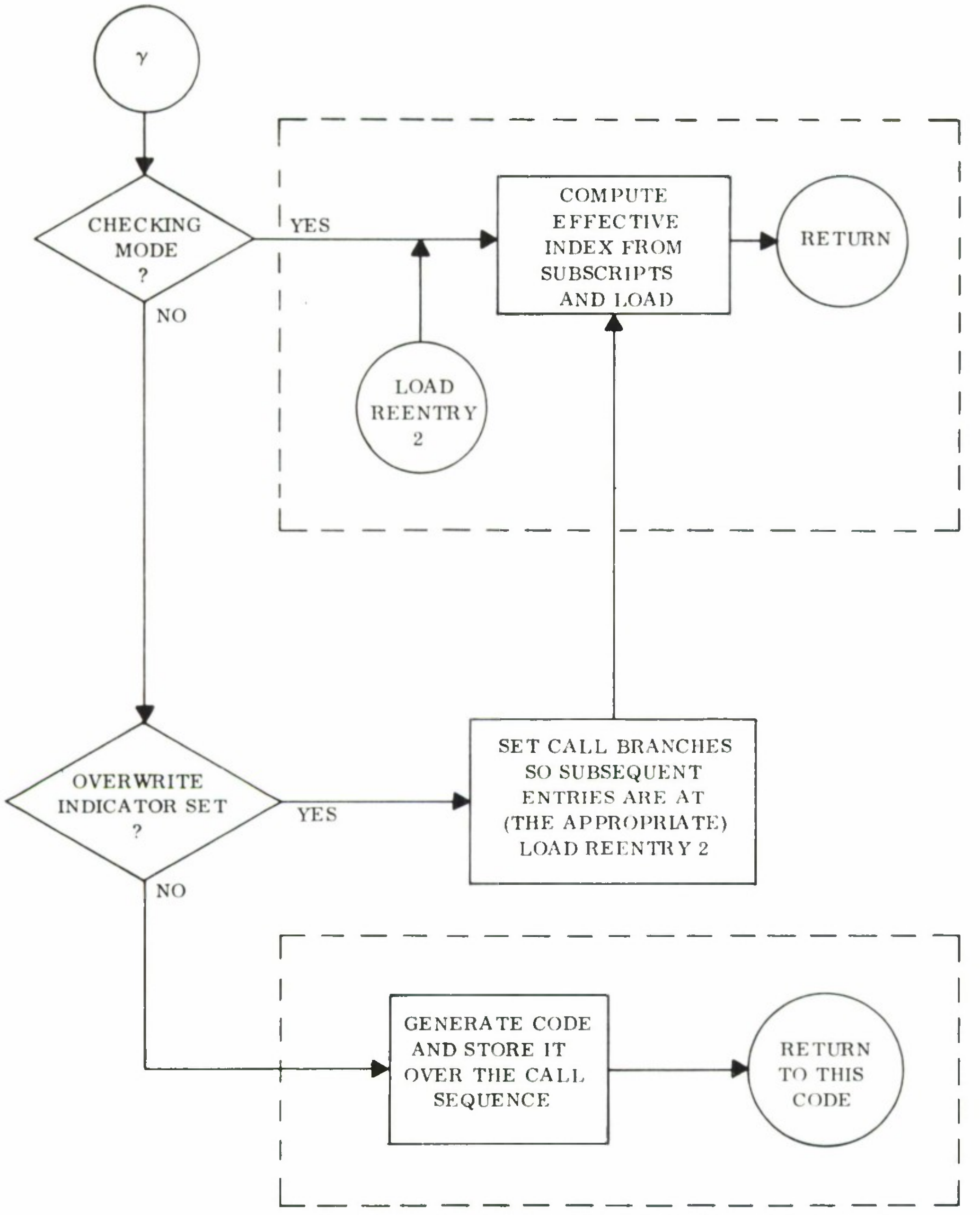
```
CALL SW (X,KW,I1,I2,...,INDIM)
```

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
X	Variable	---	The variable whose value is to be swapped with that of the subscripted block cell.
KW	Variable	See F. (6)(A)	The key word for the block.
II	Fixed Point Expression	$\emptyset < II \leq NI$	Ith subscript for the cell whose value is to be swapped with that of X.



FL IL LL
SET FOR 1st CALL PARAMETER RESET OVERWRITE INDICATOR.
IS THIS CALL PARAMETER BEING OVERWRITTEN ?
YES
NO
SET OVERWRITE INDICATOR
IS THIS THE LAST CALL PARAM. ?
NO
YES
SET FOR NEXT CALL PARAM.
ERRMESS (23)
NO
SCW PATTERN OK RESERVED, MATCHES KW ?
YES
IS KW PATTERN OK ?
YES
NO
SET CALL BRANCH SO SUBSEQUENT ENTRIES ARE AT LOAD REENTRY 1
DOES THE CALL HAVE THE RIGHT # SUBSCRIPTS ?
YES
NO
RESTORE OVERWRITE INDICATOR
ERRMESS (22)
LOAD REENTRY 1
ERRMESS (21)
Ø
NDIM
≥3
1 → i
i+1 → i
1
2
YES
Ø < I_i ≤ Ni
YES
i < NDIM ?
YES
NO
NO
Ø < I_1 ≤ N_1 and Ø < I_2 ≤ N_2 ?
NO
ERRMESS (24)
Ø < I ≤ N ?
NO
YES
γ

Figure 11. Functional Flow Chart – FL, IL, LL


γ
CHECKING MODE ?
YES
NO
COMPUTE EFFECTIVE INDEX FROM SUBSCRIPTS AND LOAD
RETURN
LOAD REENTRY 2
OVERWRITE INDICATOR SET ?
YES
NO
SET CALL BRANCHES SO SUBSEQUENT ENTRIES ARE AT (THE APPROPRIATE) LOAD REENTRY 2
GENERATE CODE AND STORE IT OVER THE CALL SEQUENCE
RETURN TO THIS CODE

Note: dotted lines enclose sections that are actually repeated four times--once each for the cases NDIM = Ø, 1, 2 and ≥3.

Figure 12. Functional Flow Chart - FL, IL, LL

(5) Operation (see Figures 13 and 14)

Operation is the same as that of ST [see Paragraph I] , except that "store" should be read "swap." Also, the comparison with FORTRAN, which has no swap operation, does not apply.

(6) Comments and Examples

(A) All comments for ST [see Paragraph I] apply with obvious modifications.

(B) It might be noted that, as is required for logical consistency, the subscripts are used to compute the address of the cell before the swap is performed; hence the statement:

CALL SW(I, IX, I, J)

causes

I to assume the value of $IX_{\text{old I, J}}$

$IX_{\text{old I, J}}$ to assume the value of old I.

L. RLSTO - Release Storage

CALL RLSTO (KW1, KW2, . . .)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
KW1, . . .	Variable	See F. (6)(A)	The key words for blocks to be released. There must be at least one such key word.

(5) Operation (see Figure 15)

The following sequence is repeated for each key word in the sequence:

(A) The associated block is located. It must be currently allocated.

(B) The key word is set to Ø and hence becomes invalid for any XPSTC associated operation except another allocation request.

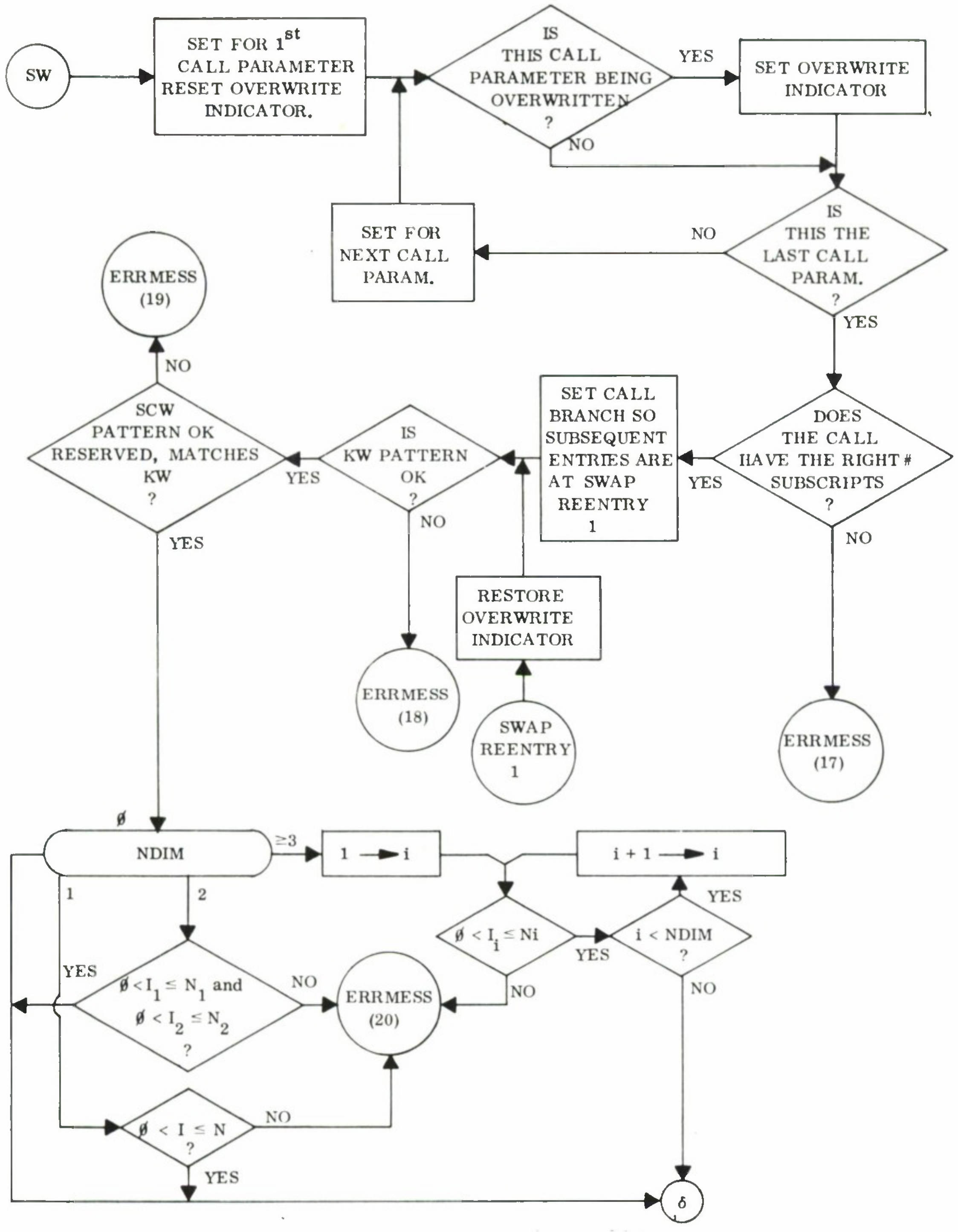
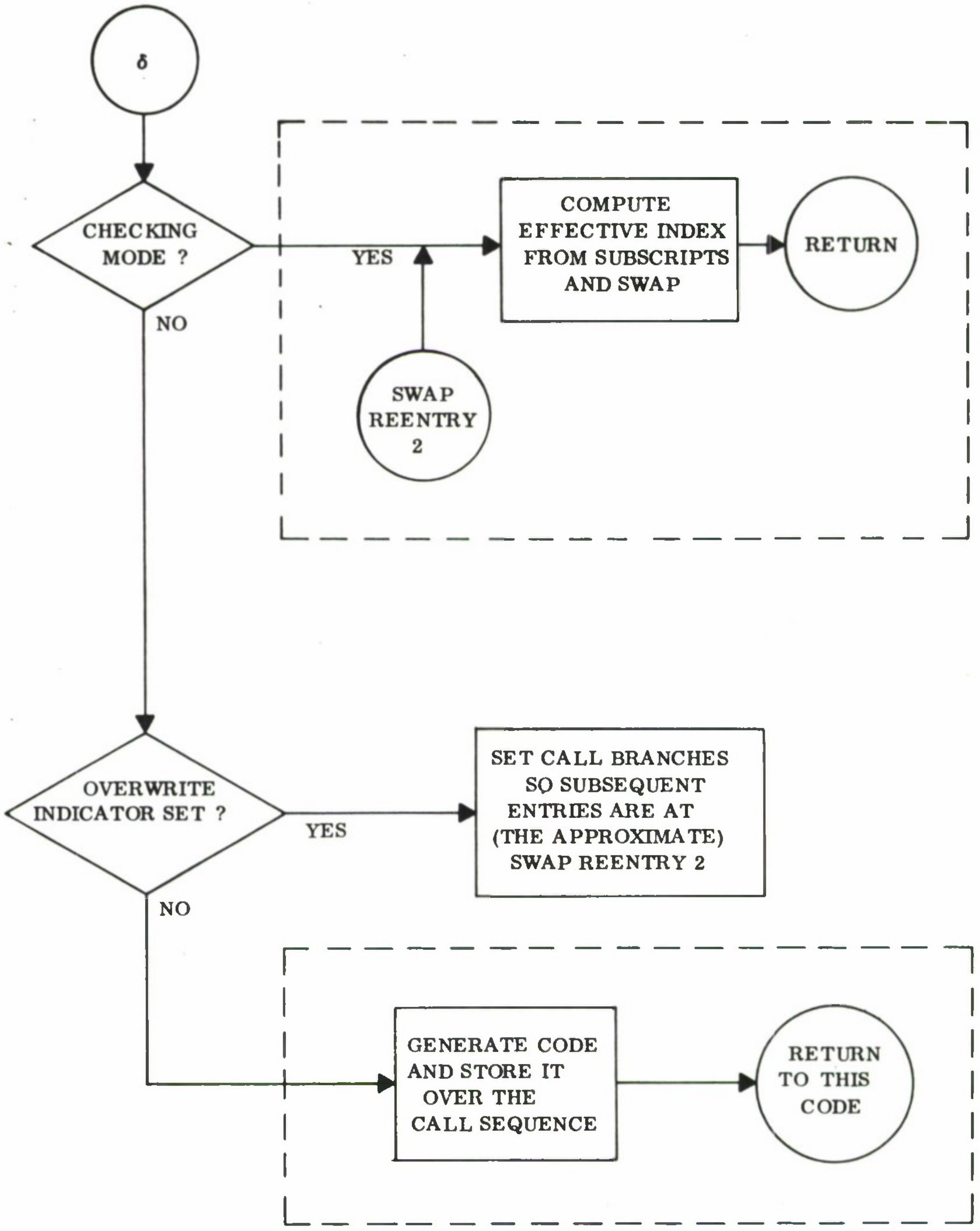

SW
SET FOR 1st CALL PARAMETER RESET OVERWRITE INDICATOR.
IS THIS CALL PARAMETER BEING OVERWRITTEN ?
YES
NO
SET OVERWRITE INDICATOR
IS THIS THE LAST CALL PARAM. ?
NO
YES
SET FOR NEXT CALL PARAM.
DOES THE CALL HAVE THE RIGHT # SUBSCRIPTS ?
YES
NO
ERRMESS (17)
SET CALL BRANCH SO SUBSEQUENT ENTRIES ARE AT SWAP REENTRY 1
IS KW PATTERN OK ?
YES
NO
ERRMESS (18)
RESTORE OVERWRITE INDICATOR
SWAP REENTRY 1
SCW PATTERN OK RESERVED, MATCHES KW ?
NO
YES
ERRMESS (19)
NDIM
ø
1
2
≥3
1 → i
i + 1 → i
ø < I_i ≤ Ni
YES
NO
i < NDIM ?
YES
NO
ø < I_1 ≤ N_1 and ø < I_2 ≤ N_2 ?
YES
NO
ERRMESS (20)
ø < I ≤ N ?
NO
YES
δ

Figure 13. Functional Flow Chart - SW


δ
CHECKING MODE ?
YES
NO
COMPUTE EFFECTIVE INDEX FROM SUBSCRIPTS AND SWAP
RETURN
SWAP REENTRY 2
OVERWRITE INDICATOR SET ?
YES
NO
SET CALL BRANCHES SO SUBSEQUENT ENTRIES ARE AT (THE APPROXIMATE) SWAP REENTRY 2
GENERATE CODE AND STORE IT OVER THE CALL SEQUENCE
RETURN TO THIS CODE

Note: dotted lines enclose sections that are actually repeated four times--once each for the cases NDIM = Ø, 1, 2 and ≥ 3.

Figure 14. Functional Flow Chart - SW

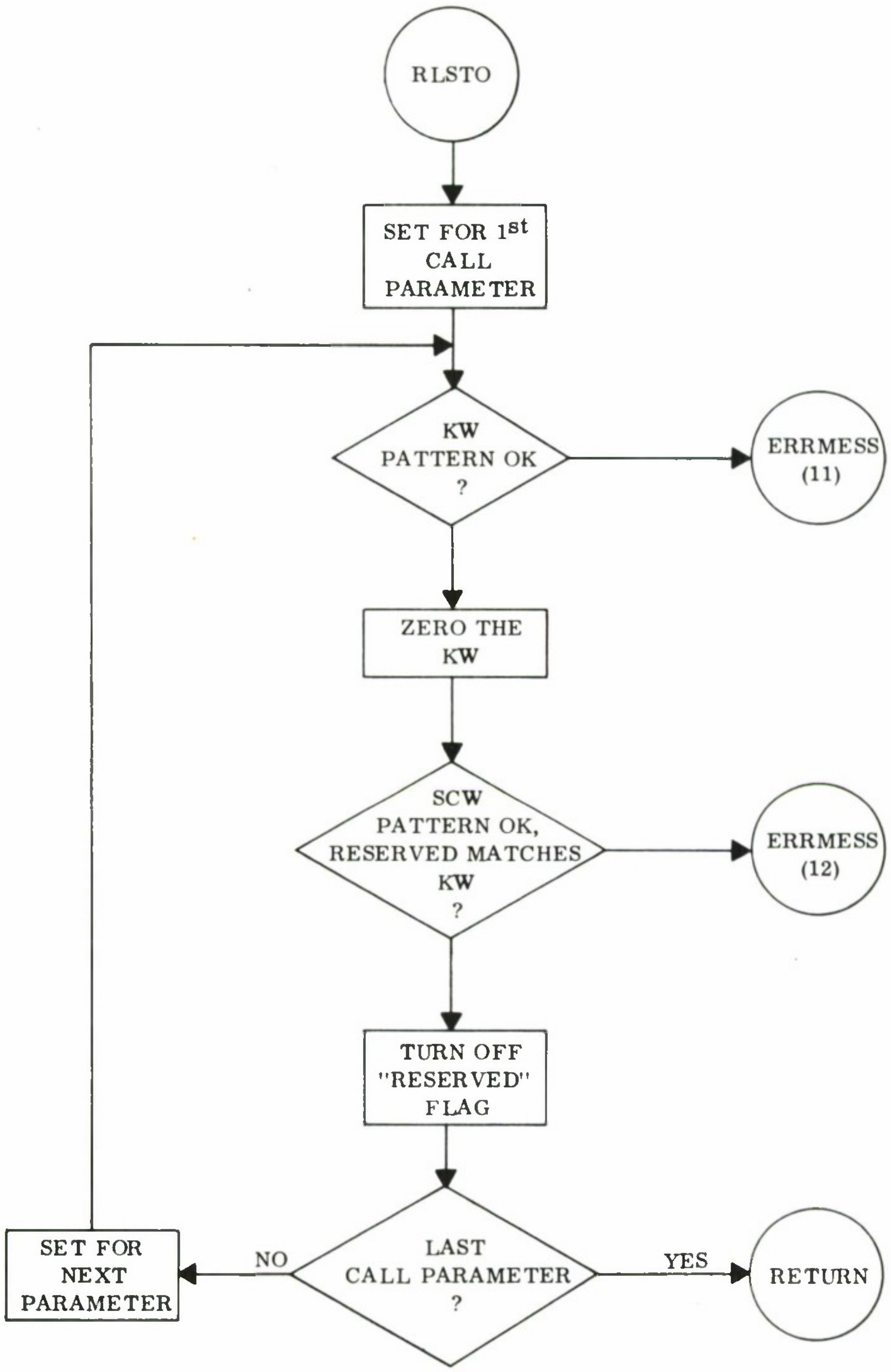


Figure 15. Functional Flow Chart – RLSTO

(C) The block is flagged "free" and becomes eligible for re-allocation in whole or in part or for overwriting during a Scratchpad packing operation.

(6) Comments and Examples

(A) Once a block has been released, it is generally not certain that the same area of Scratchpad will be allocated the next time a storage request is made, even if such a request immediately followed the release. Hence no attempt should be made to use data from a block which has been released since the data was stored there.

(B) Example: CALL RLSTO (X(3), V, K)

M. ALLOC - Allocation Status Test

Logical Function: ALLOC(KW)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
KW	Variable	---	KW of the block whose current status is being tested.

(5) Operation (see Figure 16)

If the block associated with KW is currently legally allocated, the function value is . TRUE.. If no allocation has been made, or the block has been released, or the KW or SCW have been contaminated, the function is . FALSE..

(6) Comments and Examples

(A) Under just the right circumstances, ALLOC could generate an invalid address (AD) interrupt on a legal call unless the following precautions are taken with any key words to be tested by ALLOC:

(1) A DATA KW(∅) should be used to insure the initial state of the KW at load time.

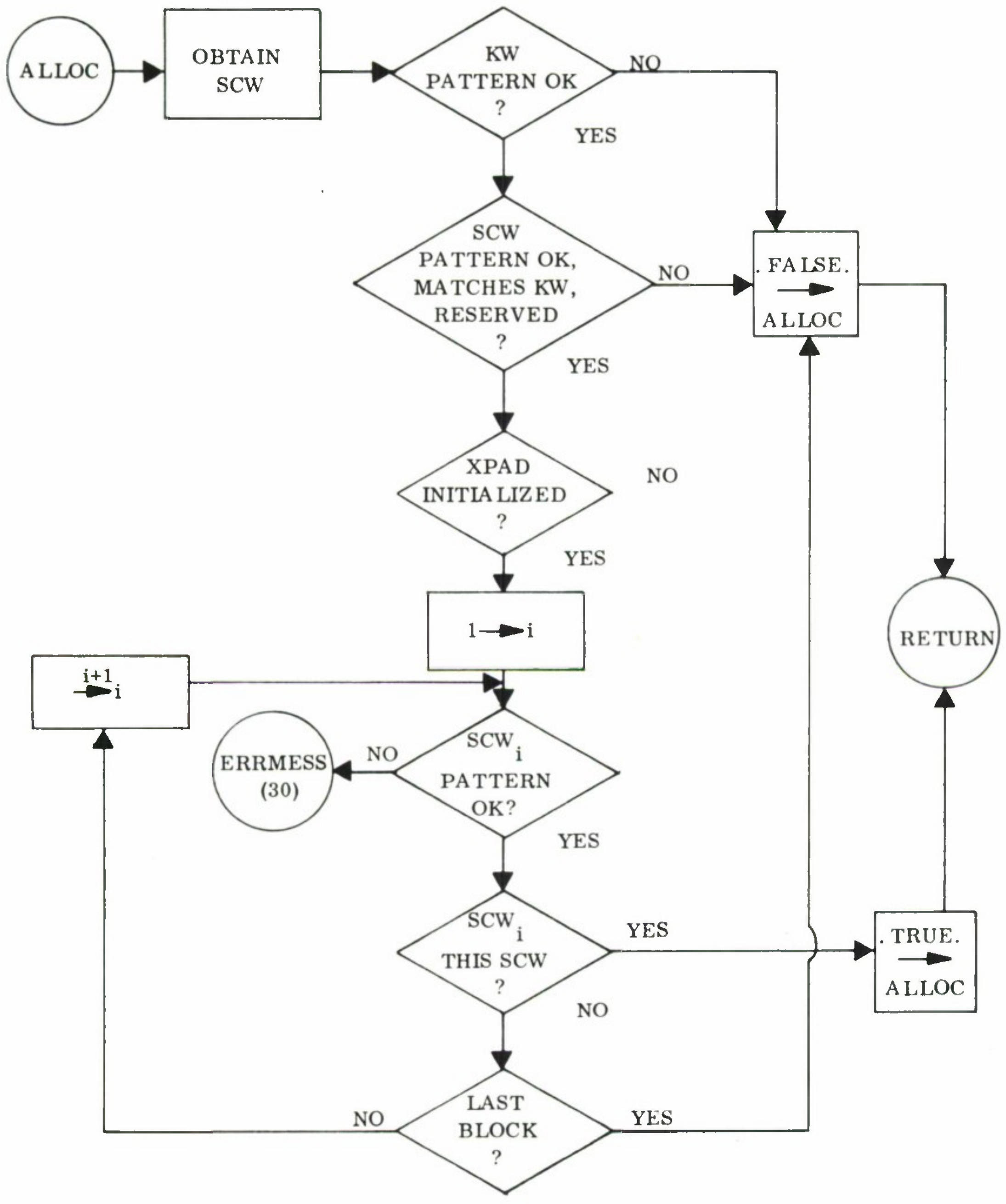


Figure 16. Functional Flow Chart - ALLOC

(2) The key word should not be used for other purposes after releasing the block unless a KW=Ø is executed after such use.

(B) In general, this function is useful only in rare circumstances.

(C) Example: IF(.NOT.ALLOC(A))CALL RQSTO(A,1Ø)

N. <u>ENTCM - Enter Checking Mode</u>

CALL ENTCM

(1) <u>Operation</u> (see Figure 17)

(A) Checking mode is established.

(B) The CM level counter, as shown below, is incremented by 1, and a return is made.

(2) <u>Comments and Examples</u>

ENTCM and the associated RESCM [see Paragraph O] may be thought of as left and right parentheses, respectively, so that the state of the system during a series of ENTCM/RESCM calls is CM until the RESCM associated with the first ENTCM is executed:

```
.........(...(..)..(..(..)))...........
As initialized  >/ checking mode  >/ As initialized  >
by INSTO [a]
```

The level counters indicate the current depth of successive ENTCMs.

O. <u>RESCM - Resume Prior Checking Mode</u>

CALL RESCM

(1) <u>Operation</u> (see Figure 17)

(A) The CM level counter is decremented by 1. If this brings the level below zero, an error stop results.

(B) If this reduces the level to zero, CM as initially set up by INSTO is re-established. Otherwise CM is left in the "on" condition and the subroutine returns.

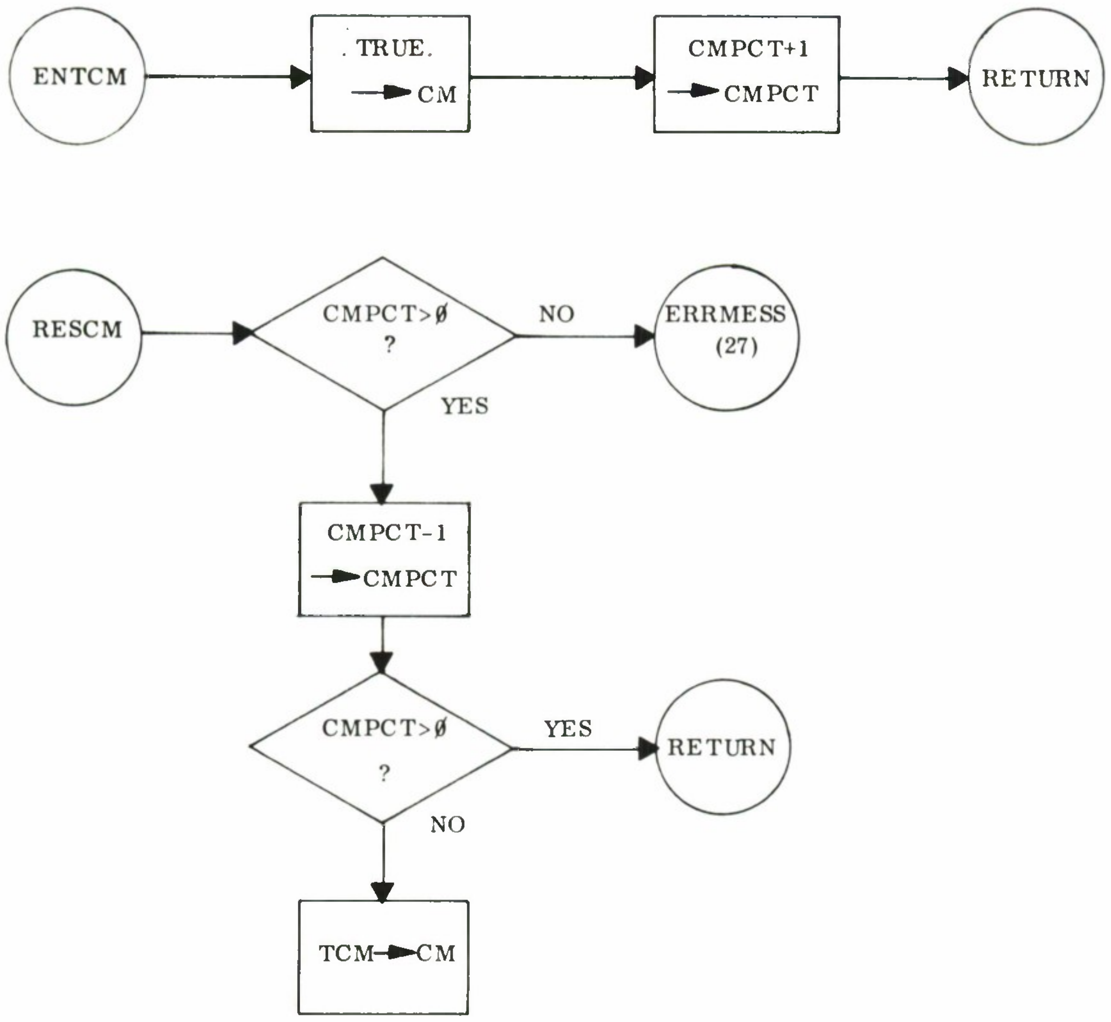


Figure 17. Functional Flow Chart - ENTCM, RESCM

(2) Comments and Examples

(A) As is implied by Paragraph (1)(A), it is considered an error when more RESCMs have been executed than ENTCMs since the last call to INSTO.

(B) Temporary establishment of checking mode has two principal uses:

(1) debugging only of a portion of a system so that checked-out segments are not slowed down by validity checking, and

(2) permanent validity checking, in places where timing is not critical and the program author cannot otherwise guarantee proper subscripts, such as:

```
      .
      .
      .
      .
READ 1, X, I, J,
CALL ENTCM
CALL ST (X,A,I,J)
CALL RESCM
```

P. ENTSM - Enter Stationary Mode

CALL ENTSM

(1) Operation (see Figure 18)

(A) Stationary mode is established.

(B) The SM level counter is incremented by 1, and a return is made.

(2) Comments and Examples

The same comments and the example given for ENTCM [see Paragraph N. (2)] hold, with obvious modifications. Note that in the case of SM, "as initialized by INSTO" is always nonstationary mode.

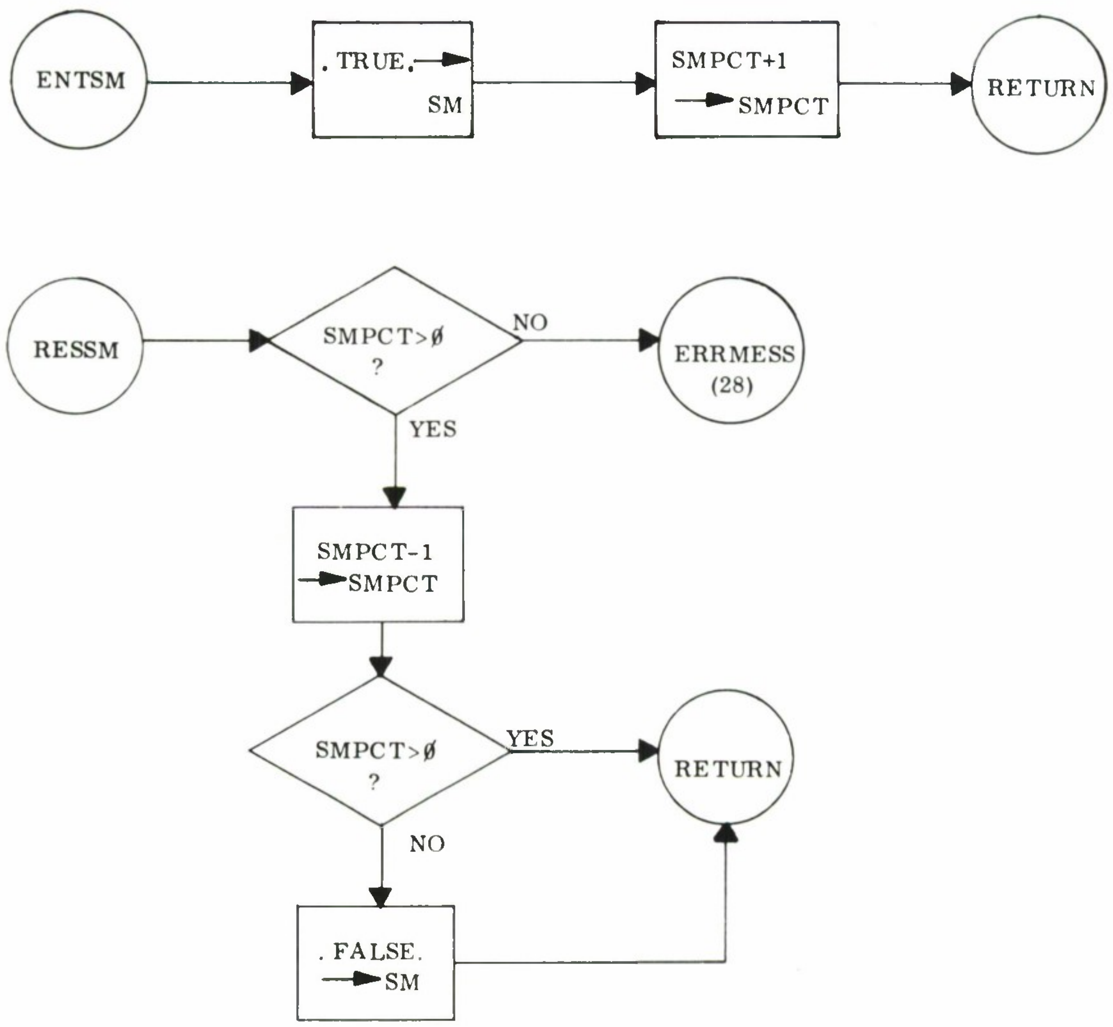


Figure 18. Functional Flow Chart - ENTCM, RESSM

Q. RESSM - Resume Prior Stationary Mode

CALL RESSM

(1) Operation: (see Figure 18)

Operation is similar to that for RESCM [see Paragraph O. (1)] except that when the level counter is reduced to zero, SM is always turned "off."

(2) Comments and Examples:

(A) Comment [see Paragraph O. (2)(A)] applies for ENTSM/RESSM also.

(B) Stationary mode is generally to be avoided except where necessary in conjunction with IBASE [see Paragraph H.]

(C) For an example of use [see Paragraph H. (6)]

R. RKW - Reassign Key Word

CALL RKW (OKW, NKW)

(1) Parameter	(2) Type	(3) Restrictions	(4) Description
OKW	Variable	see F. (6) (A)	Current key word of a block.
NKW	Variable	Not in XPAD	Word that becomes the key word for this block.

(5) Operation: (see Figure 19)

(A) The block associated with OKW is located. It must be currently allocated.

(B) OKW is disabled and becomes ineligible for further use as a key word.

(C) NKW is established as the key word for the block.

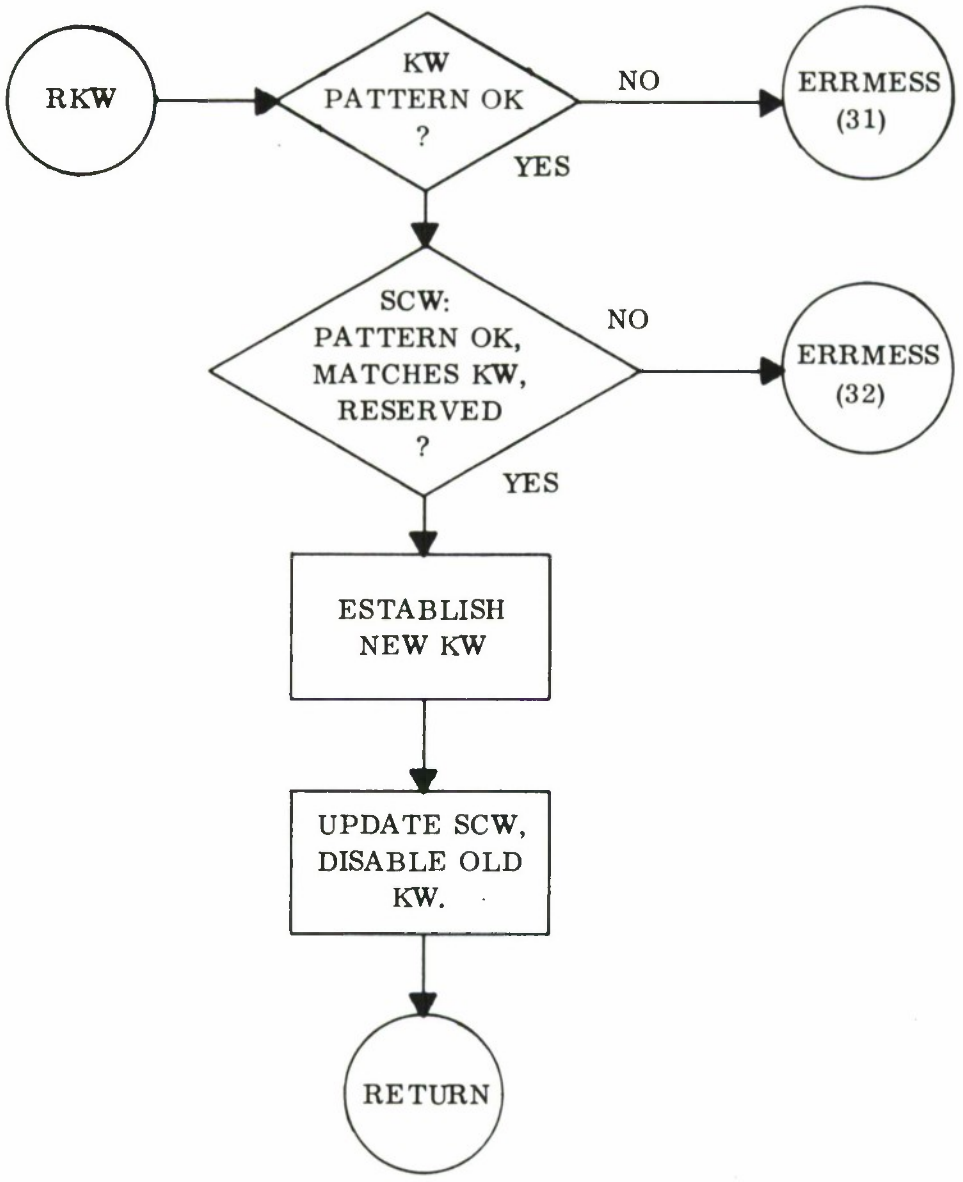


Figure 19. Functional Flow Chart – RKW

(6) Comments and Examples:

(A) When key words are arranged in a static array, it is frequently more efficient [see Paragraph III, Storing, Loading, and Swapping, page] to redefine the desired key word, using a single cell at the beginning of a long loop through the array,

```
        DO 1ØØ  I=1, NPLANE
        CALL RKW  (KW(I), TEMPKW)
          .
          .
          .
          .
        CALL ST  (X, TEMPKW, J, K)
1ØØ     CALL RKW (TEMPKW, KW(I))
```

(B) Another example is given [see Paragraph F. (6)].

S. RQXMP - Request Scratchpad Map

CALL RQXMP

(1) Operation: (see Figure 20)

(A) A map of Scratchpad, as exemplified below, is prepared and output displayed on $PRINTER via IOCS. Absolutely no checking is performed on the validity of the preface and key words encountered. No XPSTC error indication can ensue from a CALL RQXMP.

(B) All fields, unless otherwise labeled, are octal.

(C) The address before "CALL RQXMP" is the address of the branch from the user's program (or, from the XPSTC error message routine) to RQXMP.

(D) The first field is the address of the storage control word for the block.

(E) The next six columns represent the contents of this word as follows:

Key word address
Pattern bits (should be 152)
Reserve bit:
  Ø=block is free
  1=block is allocated

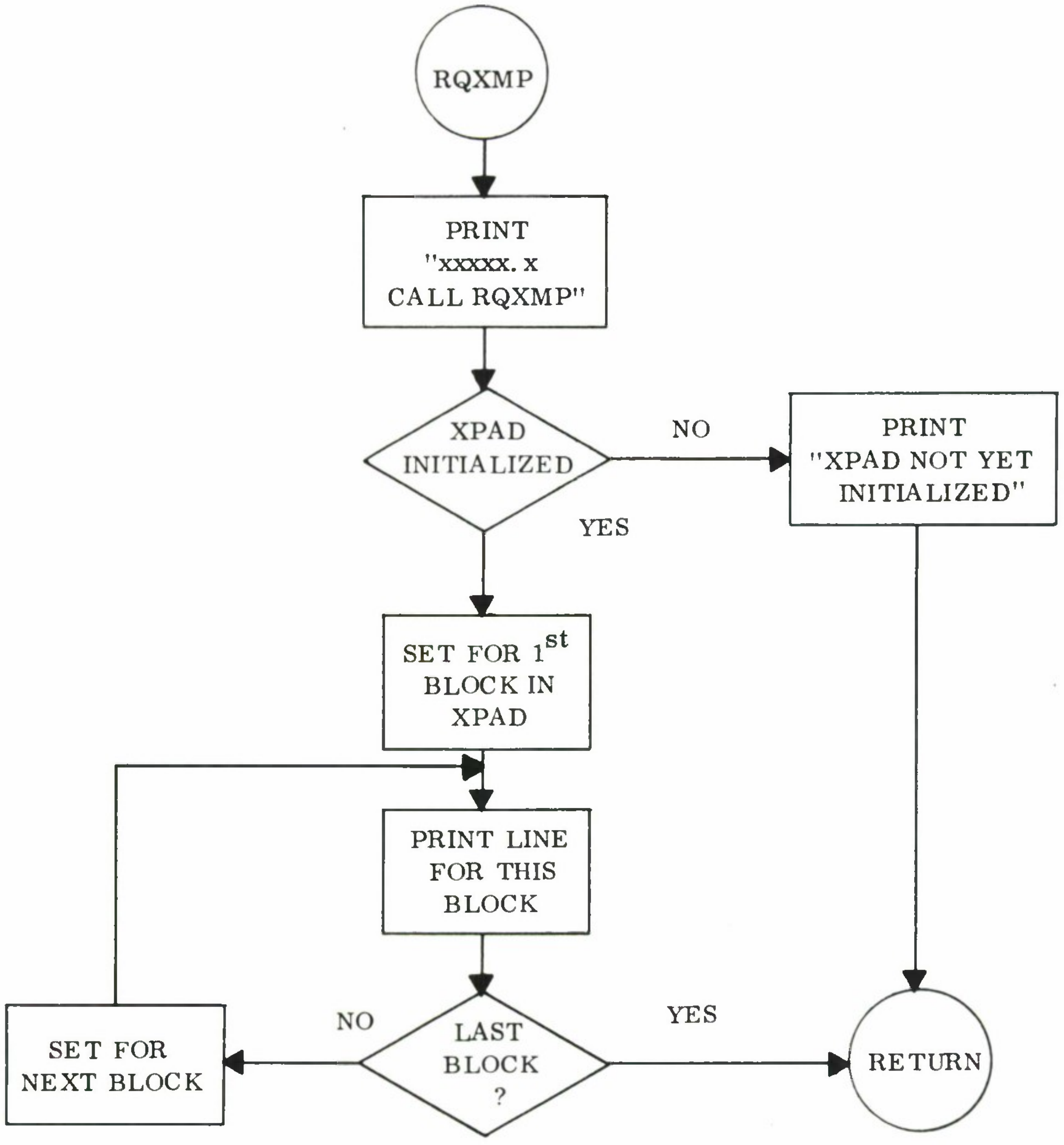

RQXMP
PRINT "xxxxx. x CALL RQXMP"
XPAD INITIALIZED
NO
YES
PRINT "XPAD NOT YET INITIALIZED"
SET FOR 1st BLOCK IN XPAD
PRINT LINE FOR THIS BLOCK
LAST BLOCK ?
NO
YES
SET FOR NEXT BLOCK
RETURN

Figure 20. Functional Flow Chart - RQXMP

Last - block bit
    ∅=all blocks but last
    1=last block only
Number of dimensions
Size of block (including control words)

(F) The next two columns represent the contents of the key word:

Storage control word address

Pattern bits (should be 1∅∅2∅32655 if the block is reserved).

(G) The next two columns represent the contents of the name word:

Name of the block, if any, may be "hash" for certain free blocks.

Type of data

∅ Floating point or unspecified
1 Alphabetic
2 Integer (Fixed point)
3 Logical

(H) The remaining fields contain the block dimensions.

(2) Comments and Examples:

(A) It is useful to call RQXMP just prior to any snapshot dump during execution or before an abnormal end-of-job (FORTRAN "STOP"). The user will find it easier to locate his data with a map of Scratchpad. Typical RQXMP output data sample is shown in Figure 21.

(B) RQXMP is also called when an error condition is detected by XPSTC [see Error Diagnostics, page 49].

```
    3357.4  CALL RCXMP

SCWAD     **(SCW)= KEY   AD   PAT  R  L   NDIM(10)  NSBLK(10)**    **(KEY)=  SCW   AD    PAT       **      NAME       TYPE      DIMS(10)
142040              2323      152  1  0       0          3                 142040       1002032655                  0
142043              2327      152  1  0       0          3                 142043       1002032655     E            0
142046              2330      152  1  0       1         12                 142046       1002032655     VECTOR       1
142062              2331      152  1  0       2         16                 142062       1002032655     ARRAY        2          3      4
142102              2334      152  1  0       3         30                 142102       1002032655     0            3          2      3  4

142140                 0      152  0  0       1          5                      0                0                  0
142145              2327      152  0  0       4        287                 142043       1002032655                  0          5      7  2
                                                                                                                               4
142604              2325      152  0  0       2         84                      0                0                  0          4     20
142730              2324      152  1  1       1         40                 142730       1002032655                  0
```

Figure 21. Typical RQSMP Output Sample

T. RQXDP - Request Scratchpad Dump

CALL RQXDP

(1) Operation: (see Figure 22)

A zero-locator dump [1]* of Scratchpad is output on the System Printer.

(2) Comments:

The prime application for RQXDP is debugging XPSTC itself. Consequently, it is seldom used.

ERROR DIAGNOSTICS (See Figures 23 and 24)

Because the allocation and use of storage are basic functions, XPSTC considers all errors serious and terminates the job abnormally after printing

```
**XPSTC DIAGNOSTIC NO. XX. THE CALL TO
 XPSTC WAS FROM XXXXXX. X(8). THE JOB
       WILL BE TERMINATED
```

and a Scratchpad map [see Paragraph S, page 45]. The call location is the address of the branch to the XPSTC subroutine which detected the error. The diagnostic number may be used in conjunction with Table II to determine the probable cause of the error. It should be emphasized that just the right kind of contamination of the key or preface words could cause certain of the diagnostics which do not list this as a probable cause.

TABLE II
XPSTC Diagnostic List

No.	Subprogram(s)	Probable Cause
1	INSTO	Contaminated SCW in XPAD
2	INSTO	Contaminated KW
3	INSTO	NSXPAD less than 3
4	(T)RQ(N)STO*	(T)RQ(N)STO* called before INSTO
5	(T)RQ(N)STO*	Contaminated SCW in XPAD
6	RQ(N)STO*	Insufficient available free space
7	ALSTO	ALSTO called before INSTO
8	ALSTO	Contaminated SCW in XPAD
9	IBASE	Referenced KW contaminated
10	IBASE	SCW contaminated, freed or does not match KW

*See Ref. 1; I. C. 6. 1

No.	Subprogram(s)	Probable Cause
11	RLSTO	Referenced KW contaminated
12	RLSTO	SCW contaminated, freed or does not match KW
13	ST	Referenced KW contaminated
14	ST	SCW contaminated, freed or does not match KW
15	ST	Wrong number of subscripts ($\neq$ NDIM)
16	ST	A subscript is too large or less than 1
17	SW	Wrong number of subscripts
18	SW	Referenced KW contaminated
19	SW	SCW contaminated, freed or does not match KW
20	SW	A subscript is too large or less than 1
21	FL, IL, LL	Wrong number of subscripts
22	FL, IL, LL	Referenced KW contaminated
23	FL, IL, LL	SCW contaminated, freed or does not match KW
24	FL, IL, LL	A subscript is too large or less than 1
25	RQNSTO	Too many or no name characters
26	RQNSTO	Illegal code for data type ($\neq$F, A, I or L)
27	RESCM	More "RESCMs" than "ENTCMs"
28	RESSM	More "RESSMs" than "ENTSMs"
29	RQ(N)STO*, ADJD	Storage-using call requires packing but SM is on
30	ALLOC	Contaminated SCW in XPAD
31	RKW	OKW is contaminated
32	RKW	SCW contaminated, freed or does not match OKW
33	(T)ADJD*	Referenced KW contaminated, or XPAD uninitialized
34	(T)ADJD*	SCW contaminated, freed or does not match KW
35	(T)ADJD*	NDIM less than 1
36	(T)ADJD*	I less than 1 or greater than NDIM
37	(T)ADJD*	Contaminated SCW in XPAD
38	ADJD	Insufficient space for expansion
39	(T)ADJD*	New NI less than 1

* (T)RQ(N)STO denotes TRQSTO, RQNSTO or RQSTO
RQ(N)STO denotes RQNSTO or RQSTO
(T)ADJD denotes TADJD or ADJD

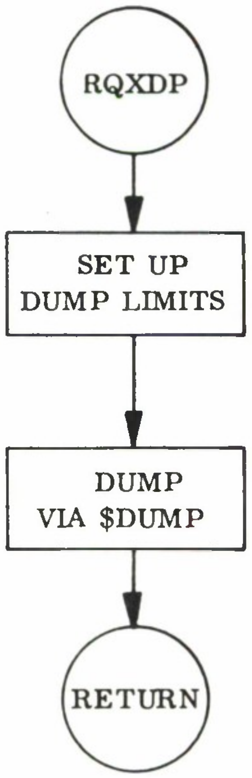


Figure 22. Functional Flow Chart - RQXDP

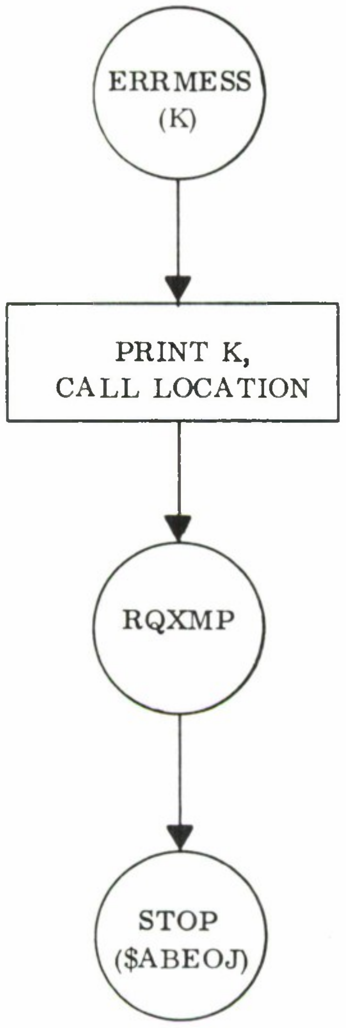


Figure 23. Functionai Flow Chart - ERRMESS (Internal Subroutine)

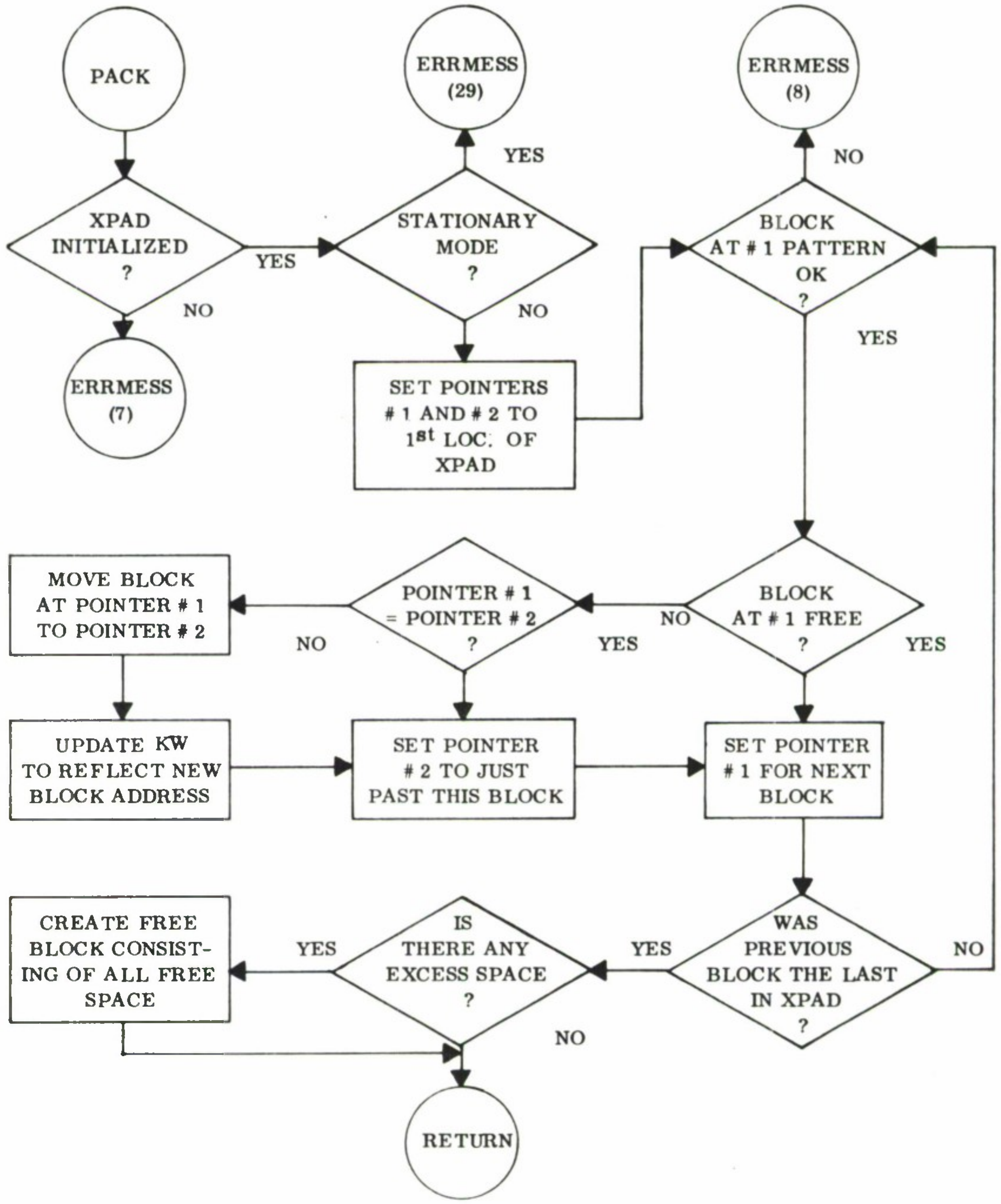

PACK
ERRMESS (29)
ERRMESS (8)
YES
NO
XPAD INITIALIZED ?
YES
STATIONARY MODE ?
BLOCK AT # 1 PATTERN OK ?
NO
NO
YES
ERRMESS (7)
SET POINTERS # 1 AND # 2 TO 1st LOC. OF XPAD
MOVE BLOCK AT POINTER # 1 TO POINTER # 2
POINTER # 1 = POINTER # 2 ?
BLOCK AT # 1 FREE ?
NO
YES
NO
YES
UPDATE KW TO REFLECT NEW BLOCK ADDRESS
SET POINTER # 2 TO JUST PAST THIS BLOCK
SET POINTER # 1 FOR NEXT BLOCK
CREATE FREE BLOCK CONSISTING OF ALL FREE SPACE
YES
IS THERE ANY EXCESS SPACE ?
YES
WAS PREVIOUS BLOCK THE LAST IN XPAD ?
NO
NO
RETURN

Figure 24. Functional Flow Chart - PACK (Internal Subroutine)

OPERATING INSTRUCTIONS

XPSTC is intended to operate on an IBM 7030 (improved version with $MR) [2]. The Master Control Program (MCP) [1]* must be in use as the monitor. XPSTC may be loaded as a binary deck in the standard FORTRAN environment manner [1]**. The subroutine IOCS*, which is required by XPSTC, is loaded automatically by MCP.

* See Ref. 1; II and 3, 22 ff

** See Ref. 1; II. B. 1. 2. 4 or 5 and Ref. 3; 27 ff

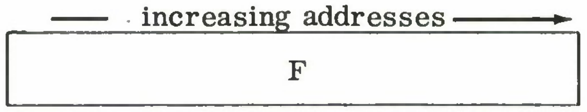
SECTION III
TECHNICAL INFORMATION

METHOD

Block Organization

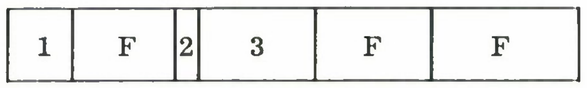
Organization within Scratchpad

At all times after the INSTO call, Scratchpad may be thought of as segmented into one or more blocks, each of which flagged as "free" or "reserved". Schematically, Scratchpad looks like this immediately after an INSTO call (F = free, n = reserved block number):



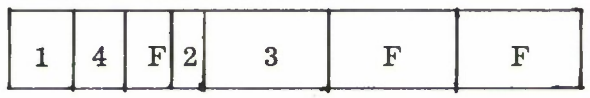
(1)

and may look like this after a series of storage requests and releases:



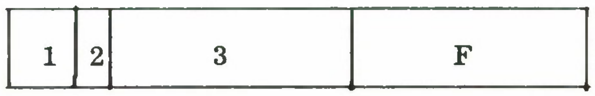
(2)

When an ordinary storage request (RQSTO) is received, XPSTC first determines the requisite extended block size [see Paragraph Internal Organization, page 57], and then searches Scratchpad for a (single) free block of sufficient size; (2) might be transformed to:



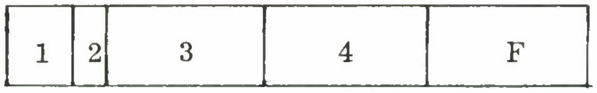
(3)

by such a call. If no such block is found, and nonstationary mode is in effect, Scratchpad is packed, (2) would become:



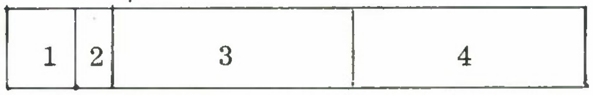
(4)

and, if possible, the block is allocated within the single free block,



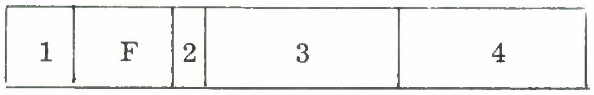
(5)

Requesting all of storage (ALSTO) will result in packing if in nonstationary mode, and allocation of the largest segment (perhaps several blocks) of continuous free memory otherwise. Starting with (2), the result would be:



(6)

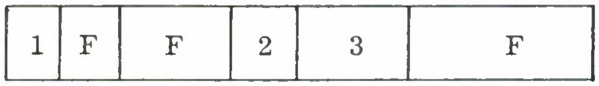
in the nonstationary mode and:



(7)

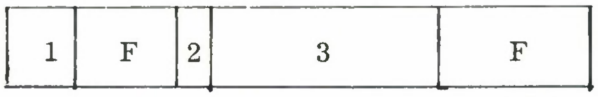
in the stationary mode.

In-flight alteration of block size (ADJD) is handled according to the type of adjustment (contract or expand). In the case where a block is made smaller, a free block is created to take up the excess. Contracting block 1 in (2) would result in:



(8)

In the case of block expansion, when free space is present immediately above the block in the amount necessary, it is used as is. Expanding block 3 in (2) might result in:



(9)

On the other hand, when such space is not present, all free space is collected above the block in question by a special type of packing (nonstationary mode only). Expanding block 2 of (2) would result in:

1	2	F	3

(10)

Internal Organization

An extended block consists of preface words (for XPSTC use), followed by the main block (available to the user). Not much need be said about the main block, except that in multidimensional arrays storage is arranged with the first subscript varying most rapidly column-wise for the two-dimensional case. FORTRAN also stores this way. The preface words occur in the following order:

SCW	(storage control word)
NW	(name and type word)
FUDGE	(present only if NDIM $\geq$3)

$$\text{NDIM}\left\{\begin{array}{l}\cdot\\\cdot\\\cdot\\\text{DWs}\\\cdot\\\cdot\\\cdot\end{array}\right.$$

(present only if NDIM$\geq$2)

The SCW is made up of the following fields:

Bits	Field
0-17	Address of the Key Word
18-24	Pattern (1101010_2,. 65 on OHEX dumps)
25	Reserved (1) or free (0)
26	Last (1) or not (0)
27	Unused
28-45	NDIM
46-63	Size of this extended block (NSBLK)

The NW has the following arrangement:

Bits	Field
0-47	Name (A8 code)
48-60	Unused
61-63	Type--Floating or unspecified (0), Alphanumeric (1), Integer (2), Logical (3)

FUDGE is in FORTRAN integer form and is present only if NDIM ≥ 3. It has the value

$$(\ldots((N_{NDIM-1}+1)\ N_{NDIM-2}+1)\ldots+1)\ N_1\ -\ NDIM$$

which is useful in subscript calculation.
The DWs, present only when NDIM≥ 2, are also in FORTRAN integer format and simply comprise a list of the block dimensions.

The key word (KW), which is not a preface word but is associated with a reserved block, has the following format:

Bits	Field
0-17	SCW address
18-24	∅
25-31	Unused
32-63	Pattern ($080835AD_{16}$) 20040. 65 AD on OHEX dumps

STORING, LOADING, AND SWAPPING

The techniques employed in these functions are best discussed with reference to the "calling sequence" compiled by FORTRAN upon encountering a CALL or Function reference with M arguments.

Schematically:

```
Y           LVI, 15, $+1
Y+.32       B, FUNCTION
Y+1.        VF, ARG1
Y+1.32      VF, SOMETHING 1
```

```
Y+2.        VF, ARG2
Y+2. 32     VF, SOMETHING 2
  .           .
  .           .
  .           .
Y+M         VF, ARGM
Y+M. 32     VF, SOMETHING M
```

In this case "FUNCTION" is ST, FL, IL, LL, or SW. The fact that "B, FUNCTION" is, in fact, through a transfer vector is of no consequence; however,any other variation of this form (loading $15 several steps prior to the branch) is illegal.

The "B, FUNCTION" as originally compiled is executed only once; once XPSTC has been reached through such a branch,the calling sequence is checked for correct length (proper number of subscripts) and the "B, FUNCTION" is replaced by a direct branch,to a point within XPSTC which by-passes this test on subsequent calls. The validity of all arguments is then checked. If XPSTC is in checking mode, the indicated function is then carried out and return linkage is performed in normal subroutine fashion.

In nonchecking mode, one of two procedures is invoked, depending ultimately upon whether or not the call sequence is being stored into. FORTRAN does not usually compile code which stores within call sequences; the exception occurs when a dimensioned variable indexed by a variable (or expression containing a variable) occurs singly (not as part of an expression) as an argument. For example:

CALL ST (SIN(X)-1., SINES, I(J+1), K*NRAD+L) (1)

falls into this category because of the I(J+1). In such an instance, FORTRAN presumes that an attempt is being made to transmit a vector beginning at I(J+1) and consequently the effective address of I(J+1) is computed just prior to the call and stored within the call sequence. FORTRAN uses the VF, SOMETHING slot immediately following the VF, ARG slot in question to effect this computation. The SOMETHING thereby becomes nonzero, whereas it is always zero in any other instance.

In an immediate sense, therefore, the decision as to procedure is based upon whether or not all the VF, SOMETHINGs are VF, Øs. For the STRAP user, this means that at least one of the VF, SOMETHINGs must be nonzero if the calling sequence is to be stored into by the calling program.

The procedure when the calling sequence is being stored into is to replace the B, FUNCTION slot once again with a branch to a point where all tests are by-passed, but the store, load or swap function is carried out and return linkage effected in normal subroutine fashion.

The procedure otherwise is to overlay the entire calling sequence with a code which directly performs the requested function; XPSTC then returns to this code and thereafter passes through the same code effect with no linkage to XPSTC whatever.

Since this last procedure is by far the more efficient, STRAP users ought to avoid storing within calls wherever possible and FORTRAN users should avoid call arguments which will cause this storage in the compiled code; statement (1) above, for example, would be more efficient as

```
        M = I(J+1)
        CALL ST(SIN(X) - 1., SINES, M, K*NRAD+L)
```

or even as

```
        IZERO = Ø
        CALL ST(SIN(X) - 1., SINES, I(J+1)+IZERO, K*NRAD+L)
```

One ramification of either procedure ought to be noted. Changes to the user's calling sequence, either the branch only or a complete overlay, are such that once a particular store, load, or swap call has been executed in the nonchecking mode, it becomes impossible to establish checking mode for that call during the same run.

CODING

XPSTC was written in the STRAP II (IBM 7030 Assembly Program) language. The coding is currently on file at The MITRE Corporation, Systems Design Laboratory, Bedford, Massachusetts.

STORAGE

XPSTC model 3 version 2, takes up 1108_{10} locations of core, which may be subdivided (see Table III).

TABLE III

Subroutine Memory Map

	Relative Loc. (8)	Size (8)	Size (10)
Transfer vector	0	2	2
ALLOC	2	24	20
ADJD	26	222	146
ALSTO	250	31.4	25.5
ENTCM	301.4	2.4	2.5
ENTSM	304	2.4	2.5
ERRMESS	306.4	53.4	43.5
IBASE	362.	21.4	17.5
INSTO	403.4	41.4	33.5
RESCM	445.0	10.0	8.
LOAD (FL, IL, LL)	455.	227	151
RESSM	704.	7	7
RQXDP	713.	10	8
RKW	723.	21.4	17.5
RLSTO	744.4	21	17
RQXMP	765.4	155.4	109.5
RQNSTO	1143.	32.4	26.5
RQSTO	1175.4	115.	77
PACK	1312.4	32.4	26.5
ST	1345	247.4	167.5
SW	1614.4	250.	168.
TADJD	2064.4	1.4	1.5
TRQSTO	2066.	1.4	1.5
Internal Storage	2067.4	34.4	28.5
		2124.	1108.

AUXILIARY ROUTINE

XPSTC calls the subroutine IOCS*, the FORTRAN I/O subroutine automatically available from the library tape when operating with MCP. [1]*

MODIFYING THE PROGRAM

Unused Module Elimination

Inasmuch as the avowed purpose of XPSTC is to use ration storage efficiently, which is neither necessary nor even particularly desirable except when available memory is limited in relation to the possible size of the data base, the rather large size of XPSTC itself (1108 locations) may appear somewhat contradictory. Actually, the trade-off is generally quite favorable in the most usual case, where a number of tables are being used and where core is not predominantly occupied by programs. When, say, 10,000 locations or more are available for data, XPSTC usurps only 10 percent or less of this space, less than is usually wasted by worst-case dimensioning within a region of this size. On the opposite extreme, however, are systems with only a few variable tables or which have but two or three thousand cells in excess of that taken up by programs. XPSTC is definitely not applicable to such situations.

A third possibility is the intermediate case, where some of XPSTC functions may be desirable but it is necessary to free the storage taken up by unused subroutines. Table IV may be used to determine the legal ways for doing this. Any module or combination of modules may be removed from the source deck, along with any associated ENTER cards, and the program reassembled. Refer to Table III to determine the current relative location of the routines indicated.

* Ref. 1; IV. D. 4. 12. 299. 13.

TABLE IV

Unit Module Chart

	Module	Storage (10)
1.	ALLOC	20.
2.	ADJD, TADJD	147.5
3.	ALSTO	25.5
4.	ENTCM	2.5
5.	ENTSM	2.5
6.	IBASE	17.5
7.	RESCM	8.
8.	LOAD (FL, IL, LL) (except card "LOAD 12")	150.5
9.	RESSM	7.
10.	RQXDP	8.
11.	RKW	17.5
12.	RLSTO	17.
13.	RQNSTO	26.5
14.	RQSTO, TRQSTO, RQNSTO	105.0
15.	ST	167.5
16.	SW	168.0
17.	TADJD	1.5
18.	TRQSTO	1.5

FORTRAN COMPILER CHANGES

XPSTC leans on a peculiarity of the FORTRAN compiler which could conceivably be changed in future FORTRAN issues. That is, FORTRAN currently signals its intention to store within a calling sequence (see Storing, Loading, and Swapping, page 58), but this is an unessential aspect of the compiler. If future

FORTRANS fail to provide such a signal, the following changes must be made to XPSTC:

(1) Card 1210 (2 instructions before "STORE3") should be changed from CMØØØØ (BU, 1), STORE 14.19 to CM1111 (BU, 1), STORE 14.19.

(2) The next card should be changed from LI (BU, 19), STORE1A to LI(BU, 19), STORE 1B.

(3) Cards 1383 - 1504 (instruction after STORE 15 through NOP after STORE3E4) may be removed.

(4) The corresponding changes to LOAD and SW should be made.

REFERENCES

(1) 7030 Facility Manual, The MITRE Corporation, Bedford, Massachusetts, August 1965.

(2) Reference Manual, 7030 Data Processing System, IBM Form A22-6530-2 (International Business Machines Corporation), White Plains, New York, 1961.

(3) Reference Manual, 7030 Data Processing System FORTRAN IV, IBM Form C22-6751, International Business Machines Corporation, White Plains, New York, 1963.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* The MITRE Corporation Bedford, Massachusetts	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE
XPSTC, A FORTRAN Dynamic Storage Allocator

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
N/A

5. AUTHOR(S) *(Last name, first name, initial)*
Sullivan, Joseph E.

6. REPORT DATE October 1966	7a. TOTAL NO. OF PAGES 77	7b. NO. OF REFS 3

8a. CONTRACT OR GRANT NO. AF19(628)-5165	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-66-94
b. PROJECT NO. 250 G	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* MTR-122
d.	

10. AVAILABILITY/LIMITATION NOTICES
Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES N/A	12. SPONSORING MILITARY ACTIVITY Deputy for Engineering and Technology Computer Programming Division Electronic Systems Division L. G. Hanscom Field, Bedford, Mass.

13. ABSTRACT

The usage and other characteristics of XPSTC, a FORTRAN-compatibility storage allocator for IBM 7030, are discussed.

DD FORM 1 JAN 64 1473

Unclassified
Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
COMPUTER						
Storage						
Floating-Boundary						
Variable						
Shared						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through
____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through
____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through
____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

Unclassified

Security Classification